目錄

Gainlo 面试指南

来源：Gainlo Mock Interview

译者：飞龙

- 在线阅读
- PDF格式
- EPUB格式
- MOBI格式
- Github

赞助我



协议

谷歌面试准备完全指南

为什么我编写本指南 凭借在这个行业的多年经验，我想给你现存的，最详细的谷歌面试准备指南。

自从 Gaino 成立以来，我困扰于如何为 Google，Facebook，亚马逊等公司准备面试。我很快意识到，大多数人对谷歌面试准备（对其他公司）没有什么了解，甚至许多人都把注意力放在了错误的事情上。

谷歌面试专门侧重于数据结构/算法，系统设计，测试等领域。有了正确的方法和资源，你可以显着提高聘用的机会。而且，你将从你应该准备的东西中受益。

因此，本资源包含了你需要了解的关于 Google 面试准备的所有信息，包括可接触的，可立即操作的事情，你可以在阅读后立即采取行动。

这个指南写给谁？

如果你正在寻找 Google，Facebook，Uber，Airbnb 等顶级公司的软件工程师职位，这是写给你的。如果你在网上被大量的编程问题困扰，那么这个指南是为你准备的，因为我会帮你筛选杂乱的东西。

虽然实习生，应届生和经验丰富的工程师有不同的标准，但 Google 面试准备的基本思路是一致的。例如，无论你经验丰富或缺乏经验，你都可以预期解决编程问题。在后面的章节中，我还将为每个类别提供实用技巧。

它是否适用于其他公司？

Google 无疑拥有最标准的面试流程，大多数科技公司也借用了类似的想法。因此，本指南适用于大多数顶级公司，如 Facebook，Uber，Dropbox，Amazon，Airbnb 等等。

当然，不同的公司有自己的重点，如文化适应。但是像数据结构/算法和系统设计这样的最重要和基本的要求在任何地方都是一样的（尽管门槛可能不同）。

一些小创业公司可能拥有完全独特的面试过程，这个资源不会涉及它。例如，一些公司会要求候选人在周末完成一个项目。

如何使用本指南？

我会建议阅读所有东西。为了节省你的时间，我尽了最大的努力把所有事情都变成可以立即采取行动的事情，以便你准确知道要做什么。

另外，这个资源假设你没有太多的时间，因为很多人只能在下班时间做准备，但是我总是相信，如果你专注于正确的事情，你会实现完美的工作/生活平衡。

即使你对这个话题有些熟悉，但是我敢肯定，你会发现一些新的武器，可以加入到你的武器库中，因为在过去的几年中情况已经发生了很大的变化。

一、获得面试机会

> 原文：Chapter 1: Get an Interview with Google
>
> 译者：飞龙
>
>
>
> 自豪地采用谷歌翻译

谷歌每年收到100多万份简历，只有少数人有面试的机会。 换句话说，大多数人没有失败于谷歌的面试，而是获得面试机会。

在谈论谷歌面试准备时，大多数人都会关注编程问题，并尝试在 Leetcode 上练习每一个问题。 但是，其中超过 90% 的人没有测试它的机会。

即使你没有做好充分的准备，也意味着你还有面试机会。 在本章中，我想对如何获得谷歌的面试机会提供许多实用的建议。 大多数技巧也适用于其他公司。

时间会有很大的变化

我们不得不承认，申请谷歌的时间可能会有很大的变化。 到了 2014 年左右，这对硅谷的科技公司来说是最好的时机。 每天都有新的创业公司诞生，每个大公司都想吸引工程师。 似乎有巨大的机会。

结果是，你比以往任何时候都更有可能获得谷歌的面试机会（与大多数其他公司一样）。 从 Gainlo 2016 年下半年的用户报告中，大多数人没有收到谷歌招聘人员的答复。 优步是另一个很好的例子，最好的时间是在 2015 年左右。

你可能无法控制何时申请谷歌，但知道时间会有很大的变化的事实，可以让你在没有得到招聘人员答复的情况下更加无忧无虑。 另外，不同的公司可能有不同的安排。 你应该时刻随机应变。

推荐是关键

我曾多次强调这一点。虽然我们有很多很棒的工具，比如 Linkedin 和在线编程平台，但推荐仍然是第一渠道。

就我个人而言，我感到很难过，因为这意味着招聘仍然采用非常古老的方式。你的工作机会仍然取决于你的关系，而不是你的技能。

因此，如果你正在准备谷歌面试，最佳方式是让你的谷歌员工朋友推荐你。说起来容易，但很难做到。大多数人不够积极，让我在这里解释一下。

首先，你的谷歌员工朋友不需要成为你的好朋友。如果他对你的工作有第一手的了解，那很好，但不是必须的。

其次，联系你的二度人脉。如果你的朋友是前谷歌员工，请问他是否可以找人来推荐你。如果你的毕业学校的人在谷歌工作，联系他。 LinkedIn 允许你轻松地检查你的二度人脉。

黄金法则是不要害怕。如果你被拒绝也并不可耻，实际上，大多数人都有求助于你的动机。研究表明，大多数人通过较弱的关系来获得工作。

你有吸引人的简历吗？

好的简历不一定会让你有面试机会，但不好的简历会绝对让你被拒绝。如果你所需要的只是几个小时来擦亮你的简历，这可能会给你一些好处，但是我看不出有什么理由你不应该这样做。

简历中最重要的两个因素是简洁易懂。

作为一个面试官/招聘人员，当我收到一份 10 页的简历时，我的第一反应就是忽略它。招聘者有数百和数千个简历可供阅读，没有人有时间阅读一本“书”。让我们简单一点，你的简历应该是一页纸。请记住，内容太多意味着，你想突出显示的内容更可能被忽略。

另外，你应该在细节和简单性之间找到一个很好的平衡点。但往往不是这样，我看到两个极端的情况。一个是简历列出了每个项目的各种技术细节与统计。如果你太过度了，人们会被搞晕。最常见的错误是提出很多公司特定的技术细节，这对公司以外的人是没有意义的。

另一个极端的例子是太简单了，有时几乎没有任何信息。 一个常见的例子是说“我构建了后端服务器”。 没有人可以评估这个项目的复杂性，因为构建后端服务器可能意味着拥有一个支持数百万个请求的复杂后端系统，它也可以意味着具有 30 行代码的简单的 Web 服务器。

那么什么是很好的平衡？ 对于没有相关背景的人，我想说你的描述应该清楚，同时给人们一些复杂的印象。 这取决于具体情况，但你可以向你的朋友询问反馈。

招聘会

如果你是学生，招聘会是你最大的优势之一。 对于大多数顶尖大学来说，谷歌和其他公司每年至少会有两次。

在网上有很多招聘会的提示，我唯一的建议是花时间准备。

当天有很多不确定因素。 例如，你不知道招聘人员会问你什么，招聘人员是否会提出编程问题等等。唯一的办法就是让自己准备好。

- 准备好你的电梯演讲
- 准备好询问每个公司的问题
- 准备好回答编程问题（不幸的是，这很常见）
- 热门公司通常有很长的队伍。 优先考虑所有你想申请的公司，并提前确定地点。

弄清楚你可以准备什么东西，什么不可以，你只需要尽早开始准备。

在线申请

你可以通过在线申请将你的简历提交给谷歌。 这是最常见和最不推荐的方法。 原因是，如果大多数人打算这样做，那么你的竞争力就会降低，除非你的简历真正突出。

不幸的是，我对这种方法的唯一建议是不要抱太大希望。

谷歌编程挑战赛

这是一个不太常见的渠道，但我仍然列在这里，以防有人满足要求。 谷歌编程挑战赛是谷歌每年主办的编程竞赛。 谷歌也试图通过这个比赛吸引工程师。

如果你已经尝试了 TopCoder，应该非常相似。 实际上，歌编程挑战赛并不是唯一的渠道。 对于大多数顶级编码竞赛，谷歌和 Facebook 等公司正在从获奖者中招募候选人。

除非你对编程竞赛超级感兴趣，否则我不建议你使用这种方法。 阅读“编程比赛对于获得工作很实用吗？”，更深入了解这个话题。 总之，获奖很难，但整个准备过程是相当有益的。

总结

如果你不断更新 LinkedIn，Github 和其他个人资料，谷歌招聘人员有可能会与你联系。 但是不要把它当成你唯一的方法，因为机会很少。

最有效的方式仍然是通过推荐。 最常见的障碍不是缺少联系，而是担心被拒绝。 试着这样思考，最坏的结果就是被拒绝了，而你什么都不会损失。 我没有看到任何东西能阻止你尝试。

顺便提一下，如果你想得到资深的面试官的更多指导，可以查看 Gainlo，以便与 Google，Facebook 等公司的工程师进行模拟面试。

二、构建扎实的基础

> 原文：Chapter 2: Build a Solid Foundation
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0
>
> 自豪地采用谷歌翻译

这是我们的“谷歌面试准备系列指南”的第二章。 如果你注意到很多工作要求，你通常会看到像“有扎实的计算机科学基础”的东西。 这实际上是什么意思？

显然，这并不意味着你有计算机科学学位，也不意味着你写了很多代码。 在现实中，扎实的计算机科学基础意味着对基础知识有清晰的认识。

在这一章中，我想深入探讨这个话题，并给你如何建立扎实的基础的实用技巧。 我可以说这是谷歌面试准备的切入点，但同时也是最重要的一步。

最短路径

很多人都在寻求谷歌面试准备的“捷径”。 如果存在捷径的话，我想说的是把重点放在计算机科学的基础上。

这里的提示是做长远规划。 最常见的错误是，清晰理解基础知识之前练习编码问题。 这些人在短期内可能会有更好的表现，但他们迟早会回来重新学习基础知识。

这是因为如果没有完全理解基本的数据结构和算法，想出正确的想法和编写无错代码几乎是不可能的。 如果你真的想做好准备，从长远的角度来看，在基本的事情上花费足够的精力将会为你节省大量的时间。

我强烈建议人们，在对基础有信心之前不要尝试面试问题。

扎实的基础对你意味着什么？

有时，人们问我，扎实的计算机科学基础是什么意思？我可以写代码，我知道冒泡排序，这是否意味着我很好？

对于编程面试来说，基础大多意味着基本数据结构和算法的清晰理解。我将在这篇文章中逐步解释这一点。但是试着回答下面的问题：

如何在图中进行搜索？每种方法的优缺点是什么？如何决定使用哪一个？ 对于对象列表，可以使用链表，数组，栈，队列和其他数据结构。你将如何决定使用哪一个？最大的优势/劣势是什么？ 动态规划和递归有什么区别？你如何比较递归解决方案和它的迭代版本？ 如果我想将我的解决方案从 `O(n^2)` 的时间复杂度提高到 `O(nlogn)` ，你会想到什么算法？ `O(n)` 又如何？

这些只是我脑海中的一些示例性问题，如果你不能立即回答所有问题，最好回顾你的算法书籍。

让我们一步一步解构计算机科学基础。

定义

首先，你应该清楚每个基本数据结构和算法的定义。 大多数人解释链表是什么没有什么问题，但它还有很多东西。

首先，一些概念有点混乱。 例如，很多候选人不完全了解动态规划和递归之间的区别。 另一个例子是栈和堆，以及它们如何在内存中使用。 这篇文章不会给你具体的答案，但如果你只进行简单的谷歌搜索，你会发现这一切。

其次，你也应该能够实现它们。 最好的例子是快速排序。 该算法在实现方面并不是很容易，但可能会非常有帮助。 如果你想实现在数组中找到第 `k` 个最大的数字，你应该用一个堆（译者注，这里原文有错误）。 BFS/DFS 也是类似的，很多面试问题都是关于如何编写 BFS/DFS（例如遍历一个树）。

优缺点

试中最大的麻烦是候选人不知道要使用哪种数据结构/算法。 根本原因是他们从来没有想过每个数据结构/算法的优缺点。 这个想法是，所有这些基本的数据结构/算法都像你的武器，如果你想在正确的时间使用正确的数据结构/算法，你应该非常熟悉他们的特性。

幸运的是，大部分资源和书籍在同一个大章节中组合了类似的工具。 例如，链表，数组，栈和队列在一个集合中。 BFS 和 DFS 在另一个集合中。

我们以 BFS 和 DFS 为例。

- 在一个树形结构中，DFS 将尝试到达最深的叶子节点，回到原处，并找到另一个叶子节点。 所以使用了一个栈，并且可以递归地（易于编写）和迭代地实现。
- 好处是，当你要检查节点是否可到达或想要找到从 A 到 B 的路径时，DFS 是最佳选择。
- 同样，BFS 横向遍历，并使用队列。
- 如果你想找到最短路径的长度或者遍历层次，你应该使用 BFS。

所以作业是针对每一个数据结构/算法，你应该问自己：有什么优点/缺点？ 我什么时候可以使用它？

复杂度

这是很多人忽略的。作为一个面试，对于我所问的每一个编程问题，我一定会问及时间和空间的复杂度。同样，在复习数据结构/算法时，你应该清楚复杂度。事实上，你也应该对你练习的每一个问题做这个。

如果你对此非常熟悉，那么当你尝试确定要使用哪种算法时，这会非常有用。例如，最常见的情况是你的解决方案很慢，面试官要求你对其进行优化。假设你有一个 `O(n^2)` 的方法，为了让它更快，我们可以这样想：

- 一般来说，为了提高速度，我们可以选择更好的数据结构/算法，或者使用更多的内存。
- 如果我们想使它成为 `O(nlogn)` ，有几种可用的工具：二分搜索，排序，BST 等等。也许你应该尝试对数组排序，看看你是否可以利用它。
- 要使用更多的内存，散列是一个选项。另外，DP 是优化递归的好方法。

如果你这样想，你的生活会更轻松，因为你有更少的选项来选择。

总结

技术面试备忘录和大 O 备忘录有很多这个主题的资源。 此外，也推荐《算法导论》这本书，虽然有些章节是可选的。

作为一名面试官，我绝对不会通过弄不清基本概念的候选人。 对我来说，这就像候选人不知道他在做什么。 相反，即使有人没有彻底解决问题，如果一切都清楚的话，他还是有机会的。

我会建议人们逐个查阅所有基本的数据结构/算法。 了解这个概念，弄清楚利弊，并自己实现。 在这一步完成之前，不要急于练习编程问题。

顺便提一下，如果你想得到资深的面试官的更多指导，可以查看 Gainlo，以便与 Google，Facebook 等公司的工程师进行模拟面试。

三、练习编程问题

> 原文：Chapter 3: Practice Coding Questions
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0
>
> 自豪地采用谷歌翻译

这是我们的“谷歌面试准备系列指南”的第三章。 谈到谷歌面试准备时，大多数人会考虑尽可能多地练习编程问题。

这部分是正确的，因为你当然需要练习。 但是，很少有人做得正确。 用正确的方法，你绝对可以巧妙地练习，并在更短的时间内得到改善。

在这一章中，我将详细解释如何用以前的实用技巧来练习编程问题。 更具体地说，你会更多地了解以下问题：

- 为什么我们需要练习编程问题
- 何时练习编程问题
- 如何巧妙地练习（这篇文章主要是关注这个）
- 推荐的资源

为什么练习编程问题

我们应该总是问自己为什么需要做一些事情。如果我们没有明确的答案，它不值得花费时间和精力。

编程面试的严峻事实是，所提出的问题与软件工程师的日常工作没有直接关系。你很少在一个实际的项目中实现一个递归算法，最常用的数据结构不是树，栈或队列，它只是数组。

然而，评估一名工程师的最具成本效益的方式仍然是编程问题，因为没有公司能够给每个候选人一个月的实习来评估他/她的技能。

这意味着，除非发明更好的面试形式，否则编程问题仍然是“考试”的主要形式。为了在这个游戏中生存，你必须习惯于那些在实际项目中永远不会遇到的问题。

不要期待再次提出同样的问题（尽管这是可能的），而是熟悉这种类型的问题，也可能对你的工作有所帮助。

什么时候

这是非常重要的，我必须在这里强调多次：在熟悉基本的数据结构/算法之前开始练习编程问题是没有意义的。

你应该检查我们以前的文章，并确保你的基础是扎实的。 第一个错误不是人们练习不够，而是过早开始练习编码问题。 这就像没有扎实的基础，一切都只是废话。

好的，在这篇文章的其余部分，我会专注于如何巧妙地进行练习。

几年前，网上没有任何资源，人们不知道该怎么准备。 现在，你可以找到你练习不到的更多的问题。 这个想法是，你不需要完成你在网上看到的每一个问题，也不需要浏览所有这些在线资源。

80/20 规则说 80% 的结果来自 20% 的原因。 我想帮助你确定你需要关注的 20%。

提示 1：编写健壮的代码

很多人都害怕亲自做一件事。 在练习编程问题时，他们所做的一切就是“用自己的想法”来解决问题。 如果他们提出了理论上的解决方案，他们认为他们已经解决了这个问题。

用我的话来说，正确的做法甚至还没有完成一半。 这是因为我见过这么多的人，即使他们的想法是完全正确的，也没有写下可靠的代码。

最常见的抱怨是“我的执行速度慢”，“我已经有了正确的方法，我只是没有得到正确的代码”，“我的代码只有很少的小错误”等等。 恕我直言，这些都不是微不足道的，如果你还记得一些东西，这不是你可以立即改掉的东西。 实质上，这些人只是没有写足够的代码而已。

统计显示，只有 10% 的程序员可以编写二分搜索而没有错误。 我希望这个数字能让你更加关注代码。

提示 2：把想法说出来

面试与考试不同，因为这是一个互动的过程。 有些候选人喜欢说“给我 10 分钟”，然后一切都沉默了。

相反，强烈建议把想法说出来。 这有很多好处：

- 展示你的沟通技巧
- 如果你不在正确的路线上，面试官更有可能纠正你
- 这可以帮助你更清楚地理解你的想法，并防止你在想法足够具体之前编写代码

但是，这个建议说起来容易，但做起来难。 大多数人习惯安静地思考。 因此，在实践中把想法说出来是很重要的。 事实上，你应该和面试一样做相同的事情。

另一种做法是与 Gaino 的朋友或有经验的面试者进行模拟面试。

提示 3：跟踪你的时间

大多数人喜欢在安全舒适的环境中练习。这是错的！请记住，如果你的实践环境与真实面试过于不同，你很可能会有意想不到的经历。

最常见的抱怨之一是“我没有足够的时间来完成代码”。老实说，我一点也不惊讶。在练习编码问题时，有多少人注意速度？很少。

我的建议很简单。如果你想改进一些东西，跟踪它。当每个人第一次跟踪时间的时候，它们都会对速度有多慢感到惊讶。人类不善于估计时间。

对于谷歌编程面试，每一个正好 45 分钟。你会打个招呼，并在第一个 5 分钟内介绍自己，最后，你可以再花 5 分钟提问。在剩余的 35 分钟内，你需要完成 2 个编程问题，其中至少有一个需要编写代码。

不要忘记，你也将与面试官讨论，所以你可能有更少的时间来编写代码。这在其他公司中是相当标准的。

提示 4：返回到基础知识

不要为了练习编程问题而练习编程问题。你需要弄清楚自己的弱点，并在选择问题时做出明智的选择。不管你多努力，总是会有更多的无法准备的问题。处理正确的问题是成功的关键。

一开始，最好涵盖许多不同类型的问题（例如链表，递归，动态规划等）。不过，以后你应该更专注。

如果你发现自己的弱点是一个特定的领域，例如树问题。通常有两种情况：

1. 你的基础不够牢固。换句话说，你对基本的数据结构/算法没有清晰的认识。在这种情况下，不管你练习了多少个问题，你的问题总是在那里。你应该做的，是回顾你的教科书，并解决根本问题。
2. 有时候，你只需要多练习。去寻找更多相同类型的问题，并把重点放在这个领域一段时间。这是一个比漫无目的的更有效的方法。

推荐的资源

你无法练习所有在线问题。 准备正确的资源可以为你节省大量的时间，这就是为什么我认为每个人都应该严格挑选在线资源。

这里有几个我推荐的资源：

- Gainlo 编程面试问题 - 我们介绍了很多带有详细分析的问题，像谷歌，优步这样的顶尖公司都会询问它们。我认为分析过程比答案更有价值。

- Leetcode - 我不建议你完成 Leetcode 的所有问题。 然而，它标出了不同层次的问题，用于很好地了解你是什么水平。
- Glassdoor - 我通常在准备的最后阶段使用这个网站。 假设你将在两周内面试谷歌。 你绝对可以在那里找到最近提出的问题。

我并不认为你真的需要 10 多本书/网站来练习。 但是，对于你解决的每一个问题，你都应该把它当成真正的面试问题。

总结

本章的重点绝对不是让你练习尽可能多的问题。你应该挑选在线资源，并在实践中聪明一些。

清楚自己的实力/弱点并做相应的准备。了解你自己而不是漫无目的地工作，这很重要。

如果我希望从本章中得到一件事情，那就是确定你的 20% 的努力，并尽可能多地关注它。

顺便提一下，如果你想得到资深的面试官的更多指导，可以查看 Gainlo，以便与 Google，Facebook 等公司的工程师进行模拟面试。

四、资深工程师、应届生和实习生

> 原文：Chapter 4: Experienced Engineers, New Grads, and Interns
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0
>
> 自豪地采用谷歌翻译

这是我们的“Google面试准备系列完全指南”的第四章。

我们的读者有着完全不同的背景。有些已经在业内一段时间了，有些是国际学生，有些正在寻找谷歌的实习。

准备过程可能会有很大的不同，取决于你的背景和目标。 这就是为什么在本章中，我想为每种面试者提供更具体的提示。

基本上，我把面试者分为三类：资深工程师，新毕业生和实习生。 如果你正在寻找管理职位，这篇文章不适合你。 另外，如果你还没有阅读以前的章节，我强烈建议你看看他们。

相似和不同

无论你的背景是什么，面试过程在内容和形式上差不多。例如，无论你经验丰富或缺乏经验，你总会被期望写出可靠的代码。

基本上，像 Google，Facebook，Uber 这样的公司的面试主要集中在数据结构，算法，系统设计和测试（我不涉及沟通这样的非技术部分）。让我把它们在这里拆解一下：

- 数据结构。候选人应该非常熟悉树，哈希表，栈/队列等所有基本数据结构。一个很好的例子是，如果你正在实现一个 BFS，你应该清楚使用哪个数据结构。
- 算法。如果面试官要求你逐层遍历树，你知道使用哪种算法吗？而且，许多人往往忽视的是时间和空间的复杂性。我想说，如果候选人在分析复杂性方面有困难，我很难给他/她一个好的分数。
- 测试。确保代码能够正常工作是非常重要的。有些人甚至是 TDD 的铁杆粉丝。在练习编码问题时，总是问自己如何测试你的解决方案？你会提供哪些测试用例？作为一名面试官，我通常会寻找能够覆盖这些角落案例的候选人，并且能够将代码模块化以便于测试。
- 系统设计。这是高级和低级工程师之间的一大区别。设计一个可扩展和健壮的系统并不是一件容易的事情。我们很快就会在整个章节中讨论这个话题。

实习生的系统设计通常是可选的。但是每个人都需要数据结构，算法和测试。

此外，即使提出同样的问题，企业对学生的期望通常也较低，这是合理的，因为他们没有太多的工作经验。

作为一个面试官，我通常会问类似的问题，不管背景如何。对于实习生，我可能会选择比较容易的问题，但是我肯定会要求大家写代码。

应届生和实习生

事实上，在面试准备方面，我并没有看到应届生和实习生之间的过大差异。 虽然实习生可能门槛更低，但面试过程是完全一样的。

专注于数据结构/算法

如果你精通基本的数据结构/算法，你至少完成了 80%。 我并不夸张。 这是因为没有人期望学生有太多的工作经验，所以公司只能测试基础知识。 另外，拥有扎实基础通常是面试官寻找的第一件事。

在我看来，如果学生对基础知识非常熟悉，那就意味着他知道自己的代码，他知道自己在做什么。

因此，在这个领域花尽可能多的时间和精力。 回顾你的教科书并练习足够的编程问题。 如果有什么捷径的话，一定要把重点放在基础上。

项目和实习

虽然面试官的期望值较低，但如果你做了一些很酷的项目，那真的会很不一样。 如果是课程项目，个人项目或实习，也没有关系。

在面试开始时，你通常有机会介绍自己，并简要解释你之前的一些“经验”。 之前的实习会很棒，因为这表明你之前从事过真正的项目。

如果距离面试还有很长的时间（半年以上），那么做一些项目，而不是仅仅做一些编程问题，确实是值得的。 另一个好处是准备系统设计面试真的很有帮助，我们很快就会在后面的章节中讨论这个问题。

另一个任务是花时间准备你的介绍。 找出你想突出显示的项目，并使你的语言清晰易懂。 不要以为面试官具有所有的背景。

资深工程师

最大的陷阱

在进行了大量的面试后，我得出结论：应届生在编程问题上总体胜过资深工程师。

每个人都知道，如果你已经在工业界工作了一段时间，你可能会发现一些编程问题棘手，难以解决。而且真的需要一些时间才能回忆起你过去所做的所有事情。

更重要的是，一旦你写了大量的生产代码，你就会有一种错觉，就是你可以很容易地破解所有的编程问题。但是，这不是事实，因为我们必须承认，编程问题与真实项目还是有很大不同的。

从这个角度来看，作为一名资深工程师，高估自己的面试技巧是非常危险的。你应该做一个新手来准备，并且脚踏实地。

以往的经验

资深工程师最大的优势之一是有以前的工作经验。但是，并不是每个人都知道如何最好地使用它。

在每次面试的开始，候选人通常有机会谈论他们过去的工作。很多人不理会这个，这是最大的错误之一。

作为一名面试官，如果候选人对新工作做了一些神奇的或非常有意义的事情，我会留下深刻的印象。换句话说，如果一些项目既不相关又不神奇，不用费神去说明。我见过这么多人不停地谈论他过去的工作，其中大部分都是无聊的。

底线是你至少应该花一些时间准备介绍。经验法则是针对不同的公司和不同的团队准备不同的故事。显示过去所做的相关事情确实是一个加分项。

时间管理

本章中我还会简要讨论时间管理，因为很多人抱怨由于作业或当前的工作，他们没有时间准备。

一般来说，学生是否有更多的时间准备，这很难说。但事实是，简短的时间表总是更好。让我详细解释一下。

有些人喜欢做一年的计划，每天只能花半小时或者没有时间。另一种方法是将你的计划挤压到三个月的时间，但每天都花费很多时间。后者通常效果更好。

对于学生来说，我知道总有课程项目或期末考试会消耗掉你的所有时间。尝试每天分配 3-4 小时。可能是早上 2 小时，晚上 2 小时。或者你可以根据你的课程调整时间表。但关键是持之以恒。如果偶尔因为某些原因而停止准备一周的话，它将不起作用。说实话，面试官不关心你的 GPA，只要它不是非常低。你应该优先考虑一切。

对于有全职工作的人，你最好利用你早晚的时间。很少有人可以提早起床，做一些工作，这是你区别于其他候选人的原因。另外，面试之前的几天，如果可能，请休息几天。

持之以恒是我在这里给出的最重要的建议。

总结

有时候，资深工程师和学生之间的差异完全在于心态。

应届生/实习生对面试太害怕了，因为他们认为他们对软件构建一无所知。 但是，由于期望低，只需要有一个扎实的计算机科学基础。

同样，资深工程师往往高估自己的面试技巧，因为他们每天都在写代码。 事实是，他们比学生更可能在编程问题上失败。

无论你的背景是什么，关注你可以改变的事情，像练习编程问题和准备你的介绍。

顺便提一下，如果你想得到资深的面试官的更多指导，可以查看 Gainlo，以便与 Google，Facebook 等公司的工程师进行模拟面试。

五、系统设计面试（第一部分）

> 原文：Chapter 5: System Design Interviews (Part I)
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0
>
> 自豪地采用谷歌翻译

这是我们的“谷歌面试准备系列指南”的第五章。

从 Gainlo 的调查来看，系统设计面试是人们所害怕的第一件事。 并不针对谷歌面试准备，而是所有的公司。

部分原因是系统设计问题通常是开放式的，所以没有标准答案。 另外，这个问题也难以准备，因为你不知道你的解决方案是否工作。

让我们在这一章解决所有这些问题。 我将简要介绍一下如何评估系统设计面试，然后提供准备和面试策略的实用技巧。

系统设计面试是什么

对于这个话题的新手，我会简单地解释一下。对于 Google，Facebook，Uber 等大多数顶尖公司来说，现场面试中至少有一个是系统设计面试。

在这个面试中，你将被要求设计一个特定的系统，并与面试官就所有细节进行激烈的讨论。但是，由于这个问题是相当开放的，面试官可以决定讨论的方向。考虑到这一点，即使是同一个问题，你可能会与不同的面试官进行完全不同的讨论。

这也是我从来不担心面试者是否曾经看到过这个问题的原因。我们以“设计一个网络爬虫”这个问题为例。作为面试官，我可以将面试集中在整个爬虫基础结构上，我可以具体讨论如何去除 URL，还可以询问如何检测页面是否已经更新。

也可能会要求你在系统设计面试中写下一些代码。但是我没有看到与一般的编程面试有太大的区别，在本章中我们不会涉及到这一部分，因为你可以参考前面的章节。

系统设计面试如何评估？

我坚信，如果你不能评估一件事情，你就不能改善它。大多数人不知道如何评估系统设计面试，他们如何做好准备？这就像你在玩游戏而不知道规则。

与编程面试不同，系统设计问题没有标准答案，因此评估过程更为主观。不过，作为一个面试官，我仍然会有一些东西。

首先，我会评估设计是否真的有效。虽然没有实现来验证，根据工作经验和一些常识，如果给出这个问题，我会问自己是否会尝试提出的方法。多数情况下，很明显的是要判断设计是否有问题，我只是用一些例子来挑战候选人。例如，如果我要求他检查某个网址是否曾经被抓取过，我会看看解决方案是否处理了 `t.co/xyz` 这样的短网址或带有 UTM 参数的网址。这是最低的要求。如果候选人不能做到这一点，我就不会深究，或者我可能会另外提出一个问题。

其次，我会检查可行性。有些候选人会提出只在理论上有效的解决方案。它可能需要无限的内存或系统是不是很复杂。无论如何，我会请他解决这个问题。验证它的一个好方法是，问自己需要多少时间和多少工程师才能实现这个设计。就个人而言，我更喜欢轻松简单的设计。如果你在一两个星期内无法制作原型，我可能会要求你简化它。

有时，候选人会想出一个复杂的解决方案，需要大量的数据和一些 ML 组件和流水线。现实中很难实现，因为这样做风险很大。你不想花一年时间在这个未经证实的想法上，那可能就是行不通的。

第三，我希望候选人清楚他在说什么。更具体地说，我想确保他知道系统应该以特定方式设计的原因，约束是什么，以及是否有其他解决方案。通常，设计问题会模糊描述。好的候选人能够告诉你什么是假设，以及如何将这个设计与其他人进行比较。为了使它更清楚，问问自己什么是替代解决方案，以及为什么以这种方式构建系统而不是其他解决方案。

在下面的部分中，我将重点介绍一些实用的技巧，并使用它们开始准备。

如何准备系统设计面试

我们不得不承认，经验胜过一切。这就是为什么一些有经验的工程师根本不需要准备。但是，你还可以做很多事情，来做出重大改变。

项目

如果你离你的面试还有一段时间（至少6个月），构建一些东西是绝对值得的。事实上每个人都可以参与宏观设计，但是只有那些真正从事细节工作的人才能把所有的事情都考虑进去。

如果你是一个学生，你可以参加一个实习，你也可以为你感兴趣的项目工作。贡献一些开源项目也是一个好主意。重要的不是要从事哪个项目，而是要从事一些工作。

我认为这很重要的原因是，没有实际工作的情况下，你不知道你的设计是否有效。有了一些实践经验，你很快就会意识到很多事情很难实现，但乍一看似乎是合理的。例如，如果你想要检查自上次抓取以来，网页内容是否已更新，并依赖 HTML 是否保持不变，则会发现许多

网页的内容相同，但注释，侧边栏等内容被改变了。这是一个我认为不适用的设计，虽然这听起来很合理。

好奇心

通常对一切都很好奇很重要。一个不错的做法是选择你每天使用的任何产品，比如 Youtube，想想你将如何从头开始设计系统。

有时产品可能会非常复杂，你也可以设计一个像 Facebook 朋友推荐的功能。如果你有时间，编写一些代码来实现一个原型是一个加分项。但重要的是，你应该试图深入细节。

尽管系统设计问题没有任何标准答案，但你仍然可以搜索如何实现这些产品/功能。将其与你自己的设计进行比较，了解其差异。强烈推荐 High Scalability（高可扩展性），但不要在特定工具上花费太多时间（请参阅“什么不重要”）。

注：一个窍门是，很多面试官喜欢问与公司有关的设计问题。例如，你更有可能在 Google 面试中设计 Google 产品/功能。情况并非总是如此，但应该重视公司的产品或类似产品。

实践

类似于编码问题，你还需要练习系统设计面试。有几种方法。你可以做一些谷歌搜索，看看别人会如何处理相同的问题，并与你的设计进行比较。例如，系统设计面试问题集是一个非常详细的常见问题分析。

更好的方法是与更有经验的人一起练习。如果你有在工业界一段时间的朋友，那太好了。向他们请求帮助。如果你不想打扰他们，你可以在 Gaino 上模拟面试。我认为互动练习总是比较好，因为整个面试过程比考试更像讨论。

什么不重要

一个常见的错误是，许多人对特定的技术过于重视。例如，他们在如何使用 AWS ，如何配置 Google 云平台以及如何使用特定的 Web 框架上花了很多时间。

我不是说这些没用，其实这些绝对是好东西。然而，从系统设计面试的角度来说，我认为面试官更关心对知识的理解而不是特定的技术。

例如，在讨论处理大数据时，作为一个面试官，我想讨论的是，如何将数据分发到多台机器上，如何将它们聚合到一起，以及如何平均分配负载。如果有人告诉我他将在 AWS 上使用 Hadoop，我会要求更多的细节，他最终还是会回答上面的所有问题。

经验法则是更多关注每个工具是如何设计的，而不是使用什么工具。

总结

系统设计面试的所有提示很难在一章内完成。我们将在下一篇文章中讨论一些现场策略。

如果本章只有一件事情，那么我希望现在就开始着手。大多数人花太多的时间来规划，但他们真正需要的是做东西。

顺便提一下，如果你想得到资深的面试官的更多指导，可以查看 Gainlo，以便与 Google，Facebook 等公司的工程师进行模拟面试。

六、系统设计面试（第二部分）

> 原文：Chapter 6: System Design Interviews (Part II)
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0
>
> 自豪地采用谷歌翻译

这是我们的“谷歌面试准备系列完整指南”的第六章。

我们将继续讨论上一章的系统设计面试。 在这篇文章中，我们将主要关注一些实际的现场策略。

假设你已经有了合理的设计（如果没有，请查看我们之前的章节），这里的想法是，如何在系统设计面试中使结果最优。 如何清晰地传达你的方法既是一门科学，也是一门艺术。 此外，许多候选人都太急于炫耀自己的知识，并倾向于使用一些在当前情况下毫无意义的流行语。 我们将在本章中解决所有这些问题和危险信号。

简单的概要设计

简单，直接和高效的系统真的会胜利。 Unix 是一个很好的例子，它用小的组件构建，每个组件只做一件事情，但是使它变得完美。我一直是这样的系统的铁杆粉丝，从系统设计面试的角度来看，它也让你的生活更轻松。

有些人真的想炫耀自己可以设计一些复杂的东西，只是从一个过于复杂的系统开始。这是完全错误的心态。

面试官关心的不是你使用的是多么酷的技术，而是你能否设计一个可行的系统。我见过这么多的候选人，在不考虑这个问题的情况下，不停地说各种流行语。你知道吗，我认为这些流行语只是废话，请在面试中忘记。

推荐的方法是从尽可能简单的事情开始，并尝试设计一个概要解决方案。如果你被要求设计 Google 自动补全系统，你最开始可以为带有指定前缀的最常见的查询提建议，以便你只需要一个日志处理器和一个建议服务器。

当然，这个解决方案在很多情况下都不起作用，比如有时我们需要个性化，数据可能会超出内存限制。那么我们可以考虑问题的优先度并逐一解决。千万不要让问题过度复杂，而让自己陷入困境。

不要赶时间

编程面试中的一个常见陷阱就是，在没有多少考虑和讨论的情况下开始编程。 系统设计面试中也有同样的问题。

请记住，没有人会期望你在几秒钟内就能设计出一个方案。 将整个面试过程视为讨论而不是考试。 如果是讨论，会鼓励你把想法说出来。 你不需要快速做出解决方案，但是你可以谈论你如何看待这个问题，你现在想要解决什么问题，以及你被什么卡住了。

作为一名面试官，我会雇佣的候选人通常会使整个面试过程非常舒适。 这完全是一个讨论过程，就像我们一起在研究这个问题。 他们不会假装知道一切。 相反，他们会一直告诉我他们在纠结什么，以及他们如何解决问题。

权衡

没有任何一种解决方案在所有情况下都能完美运行，这意味着你应该非常清楚你设计的优缺点。

请记住，你的解决方案高度依赖于限制，可以是显式的也可以是隐式的。显式的限制是面试官设定的限制，就像你只有一台机器一样。但是，大多数人不重视隐式的限制。

很多时候我们只是在不知情的情况下做出假设。揭开这些隐藏的假设可以帮助你更好地理解你的解决方案。例如，时间和空间的权衡是设计问题的共同主题。在某些时候，可以慢一点，但是可以节省很多内存。如果你有明确的理由，速度在特殊情况下并不重要，那么你的设计是合理的。

好的做法是考虑一下其他方法，以及为什么你选择的方法更好。通常情况下，更好的原因是由于一些限制和假设。所以验证这些假设是很重要的。在面试中改变你的想法是完全没问题的。事实上，这是一个好兆头，你正在考虑所有情况。

数量

很多人很疑惑，是否考虑可扩展性以及是否将数据分发到多台机器。 有时你不会知道一台机器是否足够，除非做一些数学运算。 例如，在上一个问题（Google 自动补全）中，如果你拥有所有搜索查询的粗略数量，则应该能够告诉你需要多少内存，以及将所有索引放在内存中是否可行。

换句话说，有时候无论要不要扩展，面试官都不会告诉你。 你可以通过合理的假设得出答案，并进行计算。

这是非常重要的一点，因为在眞实的项目中，优秀的工程师正在以这种方式做出很多决定。 这当然需要一些练习。

有一些现有的库

一个常见的错误就是，许多候选人都想告诉我“有些库可以做这个”，作为不详细设计这个功能的借口。

面试官是否知道现有的库？当然。但是，要求候选人设计特定的功能还有很多原因：

- 现有的库可能做得不好。例如，很多库都能够从网页中提取日期信息，但是没有一个能够完美的实现它。事实上，在很多公司公司中，这个特性本身就需要一个大的团队。
- 考虑到约束的特殊问题，可能会有更好的解决方案。
- 最后，每个设计问题都有现有的系统，但讨论这个问题还是有意义的。

但是，我并不是说在系统设计面试中，不应该使用任何现有的工具。设计问题要关注重点。如果它是一个常见的工具，或者与整体问题相比毫无意义的东西，那么使用现有的工具是完全正确的。

总结

再次说明，在系统设计面试中，经验胜过一切。 如果你对这个问题毫无经验，要想出合理的设计是不容易的。

正如我刚才所说，如果你有足够的时间，我鼓励你在闲暇时间做一些事情。

另外，我还建议与经验丰富的工程师讨论。 如果你从来没有做过这样的讨论，你怎么能期望自己在面试中表现出色呢？ 即使是本章的技巧，你也需要大量的练习，不要期望在第一次系统设计面试中掌握所有的技巧。

顺便提一下，如果你想得到资深的面试官的更多指导，可以查看 Gainlo，以便与 Google，Facebook 等公司的工程师进行模拟面试。

七、电话面试

> 原文：[Chapter 7: Phone Interviews]
>
> 译者：[飞龙]
>
> 协议：[CC BY-NC-SA 4.0]
>
> 自豪地采用[谷歌翻译]

这是我们的“谷歌面试准备系列完整指南”的第七章。

如果你正在阅读以前的章节，那么你应该已经清楚了解如何准备 Google 面试。通常情况下，如果你已经通过简历页面，第一步是电话面试。

尽管电话面试比一般的现场面试要容易一些，但大多数人都被淘汰。当然，你应该注意几个危险信号，但更重要的是，一些非常简单和实用的提示可以显着提高你的通过率。我将在本章中解决所有这些问题。

把它当作现场面试

如果你曾经参加过像谷歌，Uber，Airbnb 等许多顶级公司的面试，你很快就会意识到电话面试过程有多么相似。

一般来说，有两种类型的电话面试 - 非技术面试和技术面试。非技术性面试通常由人力资源部门进行。谈话主要是关于你的背景和激情。

技术面试是本章我们所关注的内容。除了不是面对面的交流之外，你可以期望它和现场面试完全一样。通常，你需要通过 Google Doc 等代码共享工具编写代码，有些公司甚至可能需要编译代码。

以 Google 电话面试为例。 这大概是 45 分钟，通常会问两个问题。 你一定会为其中的至少一个编写代码。

在电话面试中不太可能有系统设计问题，所以我的建议只是把它当作现场编程面试。 而已。你不需要准备特定于手机屏幕的问题。 事实上，很多面试官在电话和现场面试中都会提出同样的问题。

因此，如果你认为电话面试技术性较差且较容易，你可能需要调整你的期望。 准备好接受非常技术性的电话面试。 即使事情变得比预期更容易，这实际上是一件好事。

擦亮你的工具

职业网球运动员会尝试在眞正的比赛之前熟悉赛场。同样的道理，如果你在面试之前从未使用过代码共享工具，那么你肯定会感到不舒服，这可能会影响你的表现。

好消息是，你可以很容易知道要使用什么工具。搜索 Google 或 Glassdoor，或只是询问招聘人员。你也应该检查是否需要编译。经验法则是去除尽可能多的意想不到的事情。

下一步很清楚。像往常一样练习编程问题，但在特定工具上编写代码。有时可能需要一些时间才能适应，例如谷歌面试官喜欢分享 Google Doc，没有缩进，自动不全或高亮显示。

很多人通常都会忽略这一点，因为他们认为这是微不足道的，没用的。不过，我想说的是细节眞的很重要。干净而易读的代码可以眞正令面试官感到震惊。另外，如果你退一步，熟悉这个工具并不会占用太多的时间，它只会对你有好处。

把想法说出来

虽然我已经无数次地强调了这一点，但在电话面试中，把想法说出来更为重要。 每个人都知道手机屏幕上的沟通不太顺畅，但是很少有人眞正做出改进。

如果你认为面试是一个讨论，确保每个人都在同一个页面上是很重要的。 这里的目标不是消除沉默，而是让面试官知道你的想法。

最大的好处就是减少不必要的误会。 当我看到一个候选人沉默了一会儿时，我想知道他是否完全不知道，或者他误解了这个问题，也许他正走在正确的轨道上，几乎就在那里。 如果我知道他在想什么，当事情不正确时，我一定会提供帮助。 另外，最终的代码只是我们评估的一部分，讨论过程和候选人如何分析问题同样重要。

这里有几个例子：

- 谈谈你的整体策略。 你想从一些例子开始吗？ 你是否正在解决问题的一个简单版本？ 你打算使用动态规划吗？
- 你关心什么？ 你是否担心使用太多的内存或算法太慢？
- 你将在以后解决什么问题？ 例如，你知道当前的算法不处理一个特定的情况，提前提到这一点。
- 你卡在什么上了？ 当你发现自己没有取得进展时，说清楚你的问题。

你也会意识到，这些问题可以帮助你更好地理解你在做什么。 有时你可能会发现，自己只是在没有任何进展的情况下在白板上涂鸦，问问你自己现在想要解决什么问题。

沟通清楚

遵循上述要点，你应该更加关注电话面试中的沟通。 与面对面的面试不同，你不可以使用白板进行帮助，而声音沟通是电话面试的唯一方法。

很多同事不喜欢进行电话面试，这主要是因为沟通问题。 清楚的沟通可以使面试官有更好的体验。 这里有几个提示：

- 慢慢说，说清楚。 当人们紧张时，人们倾向于说得更快，这在手机屏幕上更为常见，因为这通常是第一次技术性面试。 通常，当我难以理解候选人的时候，给我的印象是他脑子里不清楚。 只要放慢速度，不要急于求成。 这也可以让你听起来更有信心。
- 使用速记来帮助。这是一个面对面讨论的“技巧”。不要把它写成文字稿。所有你需要的是突出重点。例如，如果你的解决方案包含三个步骤，只需列出概要。有些人关心速度。请记住，面试速度慢的人在沟通或打字方面不是很慢，但是在思考解决方案和完成代码方面却很慢（详细的讨论请查阅这篇文章）。实际上，用一些文字来减少不必要的来回讨论，最终可以节省你的时间。
- 找一个安静的环境。你会惊讶于有多少人在恶劣的环境中进行了电话面试。背景噪音，信号差和意外中断是我遇到的三件事情。有时候会让我发疯。
- 所有这些提示都不应该记住，因为在面试中你一定会忘记所有这些提示。你应该做的就是练习它们。除非你继续这样做了一段时间，否则你不会觉得自在。同样，除非你从第一天起尝试改进，否则你无法沟通清楚。

不要作弊

这是电话面试中最大的危险信号。 有些人使电话面试更像是“小组面试”。 有时我可以清楚地听到与别人的低语和讨论。 我并不认为这一点值得谈论太多，我唯一要说的是，面试官能够比预期更容易发现作弊行为。

总结

我认为本章最重要的技巧之一是你应该尽早练习沟通。 大多数人没有认识到这一点的重要性，这可能是一个让你脱颖而出的技巧。

不要指望在几天内有任何大的改进。 耐心等待，随着时间的推移，你会惊讶于你今天的水平。

顺便提一下，如果你想得到资深的面试官的更多指导，可以查看 Gainlo，以便与 Google，Facebook 等公司的工程师进行模拟面试。

八、现场面试

> 原文：Chapter 8: On-site Interviews
>
> 译者：飞龙
>
>
>
> 自豪地采用谷歌翻译

这是我们的“谷歌面试准备系列完整指南”的第八章。

在讨论电话面试之后，我将在本章中谈论现场面试。 对于大多数公司来说，这是你将要进行的“最终测试”，一旦你通过了，你会得到这个雇用。

与此同时，现场面试也是最艰苦的一步，你将在一天内有多场面试。 毫无疑问，现场面试在总体上更具挑战性，许多人真的害怕它。 我将在本章中解决所有这些问题，并像以前一样提供非常实用的技巧。

现场面试流程

没有充分理解规则，你永远无法赢得一场比赛。以 Google 为例。在那一天，你将总共进行四次技术性面试（通常是上午两个，下午两个）。在午餐时间有一个“午餐面试”，但这不是一个真正的面试，因为你没有被评估，这只是一个与 Google 员工聚会的机会。

在这四个编程面试中，至少有一个是系统设计面试。你一定会在白板上写很多代码，并与面试官进行激烈的讨论。面试形式与电话面试差不多。通常将在 45 分钟内询问两个问题，并且需要可靠的代码。所以要为非常辛苦的一天做好准备。

很多人对评估过程也很好奇。面试结束后，每位面试官都会写下一份报告，其中包含了 45 分钟的所有细节和讨论，包括你的代码。该报告可以非常详细地说明，在每个部分花费了多少时间，并且在提示之后你的反应将被包括在内。招聘委员会将根据所有这些报告进行招聘决策。

这个过程对于 Google 来说是独一无二的，而像 Facebook，Airbnb 和 Uber 等大多数顶级公司都有非常相似的流程。

电话面试 VS 现场面试

整个过程或多或少是一样的，所以你不可能对任何事情感到惊讶。 一些主要差异包括：

面对面交流意味着整个过程可以更加顺畅，但同时很多人更容易紧张。 如果你可以和你的朋友进行一些面对面的模拟面试，这可能会非常有帮助。 问题稍微困难一些。 你仍然有相同类型的编程问题，然而，最常见的误解是，它会比电话面试问题困难得多。 我会尽快详细讨论。 至少需要一场系统设计面试。

对于现场面试中我应该练什么样的问题，我只想说就像电话面试一样（系统设计面试除外）。 你并不需要找到现场面试的具体问题，事实上，在这两种情况下都会提出很多问题。

“问题比我想象的要容易得多”

这是现场面试中最常见的评论。 但是，这也是大多数人不能通过面试的原因。 让我详细解释一下。

最大的误区之一是 Google 的现场为什么比电话为什么要困难得多。 当然，有时候可能会有一些难题。 但总的来说，这只是稍微困难一点。 许多面试官仍然会问如 2-sum 的问题，来获得候选人的最初想法。

在 Gaino 上，很多用户抱怨说模拟面试太简单了，尽管他们没有通过。 他们不能相信问题是“容易的”。 但实际上，面试官正在将类似的问题用于真正的面试。

让我告诉你一件事。 谷歌面试中最困难的部分（与其他公司一样）不是提出解决方案。 很多人可以在一定程度上解决这个问题。 但要获得好成绩，你需要：

- 拿出最佳的解决方案
- 编程干净和无错的代码
- 在时间限制下完成这两件事情

第二点和第三点是最具挑战性的部分，大多数候选人由于这两个原因而失败。 许多人声称问题比他们想象的要容易得多，但他们仍然没有快速地提供无错代码。

从面试官的角度来看，提出没有人能解决的问题是没有意义的。 为了评估候选人，我们需要看到分析过程，代码以及他/她能够多快解决问题。

任务：

- 更多关注“基本”问题而不是超级难题。 只要在 Glassdoor 上对 Uber 面试问题进行一点搜索，就可以了解哪些类型的问题很受欢迎。 你很快就会意识到，他们并不像大多数人所想的那样艰难。
- 练习时要特别注意编程。 为每个问题写下可靠的代码并跟踪时间。

白板

对于大多数公司，你需要在现场面试的白板上编写代码。 就个人而言，我觉得这很不舒服，甚至与编写代码共享工具相比。

一个问题是插入非常困难。 通常情况下，你会发现你在开始时缺少一个简单的检查，你需要插入一个小的 `if` 块。 但是，你绝对不能在白板上做到这一点。 同样，如果你的代码是多余的，你不能通过复制和粘贴来节省时间。

如果你之前没有这样做，只是试着在一张纸上写代码，你很快就会发现它很糟糕。 因此，经验法则是提前练习。 是的，你需要在现场面试之前在白板上练习编码。

你可以在亚马逊上买一个非常便宜的白板，或者至少你应该在纸上练习编码。 同样，理念是尽可能接近面试，因为你不想在这一天感到惊讶。

这里有几个提示：

写之前要仔细考虑。 很难撤消你的代码，确保每一行代码都是你想要的是很重要的。 这实际上是一个非常好的做法。 许多人倾向于在思考时进行编码，这从来没有奏效。 事先要有清醒的想法。 工整的写作。 我不会推荐连笔。 作为一个面试官，我经常发现它是无法辨认的。 有些人可能会认为可以节省一些时间。 但是，你将花更多时间来解释代码。 加速的最好的策略是要有清醒的思路，确保顺利实现。

沟通

我在之前的文章中多次提到过它，我只想在这里强调一下几点。

- 整个面试是一个讨论过程，在现场面试中更是如此。这与考试完全不同，因为你需要与面试官保持沟通。把这个过程当作一个正常的讨论，与你们的同事在一起讨论同样的问题。
- 把想法说出来。谈论你脑海中的任何事情，在讨论之前你不需要有任何具体的想法。对你的话要小心谨慎，但对代码更加谨慎。大多数人采取相反的做法而没有通过面试。
- 随意谈谈你卡在了什么地方。面试官真的很乐意通过给你提示或告诉你你不是在正确的方向来帮助你。不要担心暴露你的弱点，即使你不承认也不会有任何进展。相反，放宽心态，面试过程将会更加愉快。

总结

同样，不要指望自己能够立即完成所有这些技巧。 面试前你一定需要很多练习。 换句话说，知道这些建议没有任何作用，除非你真的能遵循它们。

我不会涉及太多非技术性的问题和提示，我将用一个完整的章节来解决所有这些问题很快。敬请关注！

顺便提一下，如果你想得到资深的面试官的更多指导，可以查看 Gainlo，以便与 Google，Facebook 等公司的工程师进行模拟面试。

九、非技术问题

> 原文：Chapter 9: Non-technical Questions
>
> 译者：飞龙
>
>
>
> 自豪地采用谷歌翻译

这是我们的“谷歌面试准备系列完整指南”的第九章。

如何在编程面试中准备非技术性问题，是我收到的最常见的问题之一。 大多数人不知道该怎么准备，甚至不知道会问到什么类型的问题。

一般来说，非技术性问题只占面试的一小部分。 然而，如果面试官抓住任何危险信号，它可以“杀掉”你。 一个例子是文化适应。 趋势是公司正在更多地关注文化适应，对于像 Google，Airbnb 和初创公司这样的大公司来说都是如此。

在本章中，我将涵盖一切非技术性的内容，包括自我介绍，文化适应，向面试官提问等。这些不仅是可以帮助你做好准备的技巧，也是一个很好的做法，可以让你 知道你是否合适。

面试官在寻找什么？

如果编程面试用于评估应聘者的技术技能，面试官通过询问非技术性问题寻找什么？我将总结以下三个方面。

总体印象

最后，我们都在和人打交道。不管候选人有多强，面试官通常会自问一个简单的问题 - 我想和这个人一起工作吗？这确实是一个非常主观的问题，但通过与候选人交谈和讨论，面试官可以得到一些想法。

例如，有时我发现候选人对公司和产品非常热情，这可能会影响“得分”。

文化适应

本章稍后会讨论这个问题。但总之，每个公司都有自己的文化，评估候选人是否合适是非常重要的。每种文化都有其优点和缺点，因此都是关于匹配（就像约会一样）。

加分和危险信号

编程问题永远不能涵盖一个人的每一个方面。通过讨论过去的经验，候选人感兴趣的领域，面试官实际上是寻找任何加分或危险信号。

例如，如果你以前的经验是高度相关的，这绝对是一个优势。对于应届生来说，在一家顶级公司实习可能会给面试者留下积极的印象。另一方面，如果你在以前的公司只呆了三个月，面试官可能会问你更多的问题。

自我介绍

“介绍你自己”可能是唯一确定的问题，没有理由不准备。在好的和不好的介绍之间没有明确的界限，但事先做好一些准备工作肯定会对你有好处。事实上，这不会占用你太多的时间。

黄金法则是清晰简明。首先，列出你真正想要在你的介绍中突出显示的几点（少于三点）。这可能是你过去的项目之一，或者你赢得的一些编程比赛。其次，在 1-2 分钟内压缩所有这些。

你不需要覆盖尽可能多的细节。如果面试官想知道更多，他们会问。一个常见的错误是过度推销自己。长时间的介绍不仅会让面试官厌烦，而且会缩短你的编程问题的时间。

另一个常见的错误是我所说的“太模糊”。往往不是因为候选人沟通不好，而是因为介绍有很多细节，没有相关的背景的人不能理解。找到一个对你的工作不太了解的工程师朋友，测试他/她是否能理解你的介绍。

这里的另一个提示是相关性。强烈建议对每个公司有不同的介绍。你想表明你能胜任这个职位，因为你有相关的经验。这绝对不是必需的，但是如果你有的话会是一个加分。如果你以前的所有项目都是完全不相关的（例如，你在硬件上工作，但是在寻找一个软件的职位），那么就把它保持简洁。所有的细节都谈论一个完全不相关的项目，这不能再糟糕了。

文化适应

现在，我们来讨论一下这个趋势词“文化适应”。近几年你可能会一遍又一遍地听说文化适应，大多数公司开始比以往更关心它。我认为这不只是一个炒作。我想说每个人都应该考虑一下，这对公司和候选人都是如此。

那么文化适应什么？“文化适应是一个难以定义的概念，但每个人都知道什么时候缺少这个概念。”每个公司，无论多大，都有一定的规范和信念，每个人都遵循。这不是一个强迫人们以同样的方式行事的具体规则，也不是人们从不行动的口号。这就是为什么文化是定义它的最好的术语。

最着名的例子是 Facebook 的“行动迅速和打破常规”。核心的信念是尽早测试，来避免过度工程。你可以通过 Google 深入了解这一点。关键在于面试官有责任评估候选人是否真正符合这种文化。这并不容易，但从所有的讨论中，人们仍然可以得到一些想法。例如，在系统设计面试中，面试官肯定知道你是否倾向于过度工程。

另一个例子是 Airbnb。通常情况下，你将有一个完全集中于文化适应的简短面试。你会被问到你的产品经验，为什么你想加入 Airbnb，你也可以谈论你的职业目标。如果你不是真的相信 Airbnb 的使命，只是瞄准加薪，我怀疑你是否会得到录用。

双向选择

“如何让自己适应公司的文化？”显然是一个错误的问题。你不需要适应文化，你应该评估文化。

最大的误解是认为面试是考试。但是，我认为这是一个双向的选择。面试过程中，考生应同时对面试官和公司进行评估。在与公司工作的人交谈之后，你可能会有完全不同的印象。

在文化适应方面，我认为更好的解释是评估公司是否真的适合你。如果你不相信它的价值，你不需要显着地调整自己来适应公司。但是，你至少应该调查和了解它的文化。

很多人不了解他们正在面试的公司。对我来说，就像你在和一个女孩约会而不知道她的背景。这就是“为什么加入我们”应该是一个非常简单而直接的问题。如果你知道公司是否适合你，你应该可以在几秒钟内回答。如果你不相信公司的使命，也许你本来不应该申请。

给面试官的问题

在每次面试结束时，通常会有五分钟向面试官提问。 这个过程是完全可选的，但建议问一下。 好处是你会更加了解面试官和公司。 而且，向公司展示你的兴趣总是一个好兆头。

不要认为这个过程是增加机会的一种方式，尽管可能是。 一个更好的方法就是询问你脑海中关于公司的任何问题，而且把其他事情考虑好。

不要问你的面试表现/结果。 你不但不能得到真实的反馈，而且会让面试官感到尴尬，因为他们不能马上告诉你结果。 这对 HR 来说是一个问题，而面试官不是。

总结

尽管非技术性问题可能成为面试的重要部分，但事实是，大多数人往往会过度担心。

大多数时候，你只需要做你自己。 如果他们觉得你在文化方面不太合适，那可能不是什么坏事，因为它比加入公司好得多，一切都不像你想象的那样。

在面试之前，你绝对可以准备好非技术性的问题，但是要专注于技术性问题，这是面试的核心。

顺便提一下，如果你想得到资深的面试官的更多指导，可以查看 Gainlo，以便与 Google，Facebook 等公司的工程师进行模拟面试。

十、非谷歌的面试

> 原文：Chapter 10: "Non-Google" Interviews
>
> 译者：飞龙
>
>
>
> 自豪地采用谷歌翻译

这是我们的“Google面试准备系列完整指南”的最后一章。

如果你持续关注我们的博客，我希望你对如何准备 Google 面试非常清楚。 很多人可能会问，这个指南是否也适用于非 Google 面试。 这是本章的重点。

简短的答案是肯定的，你可以使用这个指南来准备 Facebook，Uber，Airbnb 等大多数顶级公司的面试。 这正是你看到一些人刚刚得到了所有这些公司录用的原因。

在本章中，我将讨论非 Google 面试风格，准备“捷径”以及如何针对一家公司进行准备。

惊人的相似

如果你面试了很多公司，你会发现面试过程有多相似。 更重要的是，同样的问题可以由不同的公司提出。

事后看来，这很有道理。 公司正试图发明准确高效的系统来评估软件工程师。 在这一点上，最广泛采用的方法是通用编程面试和系统设计面试。 这种机制可以让公司在一天之内获得候选人的最多信息。

当然，更好的办法是临时聘请候选人三个月，然后做出招聘决定。 但是，没有公司能负担得起。

如上所述，由于种种限制，“Google 风格面试”是迄今为止的最佳解决方案。

这对候选人来说是一个好消息。 你绝对可以花半年的时间准备面试，这对大多数公司来说都是有效的。

扎实的基础

这是我在本指南中最后一次强调计算机科学基础。 具有类似的面试流程意味着扎实的基础比以往更重要。

完全错误的策略是在第一天跳进编程问题。 更重要的是，他们会去 Glassdoor 找到某公司最近提出的问题并做好准备。 如果没有很好地掌握计算机科学的基础，这种方法真的是专注最不重要的事情。 更糟糕的是，这不仅对于那个特定的公司不起作用，它只是浪费你和所有公司的时间。

另一方面，如果你从一开始就把注意力集中在基础上，无论你正在准备哪家公司，你都会处于有利地位。 这种复合效应不能再重要了。

持之以恒

我不希望你有这样的感觉，我们的帖子可以给你一些技巧，立即增加你的机会，不费吹灰之力。 相反，我鼓励大家持之以恒。

尽早开始准备，并把整个过程当做自我提升而不是通过测试的一种方法。 正如我刚开始提到的，如果你只剩下一个星期，本指南不适合你。 我专注于帮助那些能够始终如一地努力实现目标的人们，而不是急于立即获得满足。

这个策略与我们提供的所有技巧一致。 如果你还有至少几个月的时间，去构建一些东西。 从事一个项目是提高技术技能的最好方法，你会发现在编程和系统设计面试（特别是后者）中都有价值。

总是寻找捷径，这是一种不好的心态。 如果有任何捷径，捷径就是持之以恒。

公司特定的准备

公司可能仍然有一些特定类型的面试。一个例子就是亚马逊在第一轮面试中通常会有一个编程测试。我的建议很简单 - 不要太担心。

我建议人们仍然按照他们的初步准备计划。总是需要着重于基础和练习编程问题。如果没有掌握基本的数据结构和算法，你将无法通过在线编程测试。

在面试之前（也许提前一周），你可以做一些公司特定的准备。强烈建议使用 Glassdoor，你可以找到目标公司最近提出的大量问题。换句话说，你应该在准备的最后一步做这个，或多或少你会发现公司之间的差异。

正因如此，当人们问我如何准备亚马逊的在线测试时，我建议他们只要做你需要做的事情，实质上根本没有什么不同。

值得注意的是，一些初创公司的面试风格非常不同，这个提示并不适用。例如，许多初创公司会给候选人一个真正的项目，并期望他们在一个周末或几个小时内完成。如果你一直在做很多真正的项目，这对你来说应该不难。

总结

如果我希望你得到一个秘籍，它一定是专注于“基础”。 大多数人往往目光短浅，大部分时间花在“细枝末节”上。 如果没有扎实的基础，你或许能够偶然地解决一个问题，但是你永远无法解决所有问题。

很多人告诉我，他们发现这个指南很有用，帮助他们得到了梦想中的工作，我很高兴。 如果你还有其他问题，请随时给我发一封电子邮件或在博客上发表评论。

系统设计面试问题集

系统设计面试的问题可能是相当开放的，这就是很多人都害怕这种面试的原因。尽管工作经验在系统设计面试中起到重要的作用，但并不意味着你无法做好准备。

我们 Gainlo 团队已经手动选择了一系列流行的系统设计面试问题，并提供深入的分析。如你所知，即使是同一个问题，不同的面试官也可以使面试完全不同。因此，请不要把我们的文章当作标准答案，而是要用它们来获得如何分析不同问题的大量思想，并减少你对系统设计面试的压力。

我们选择的许多问题既实用又有趣。例如，在“如何设计 Twitter 系列”文章中，我们已经涉及了广泛的主题，包括搜索，排名，推荐，可扩展性等等。我敢肯定，即使你没有准备系统设计面试，你也会从我们的文章中学到很多东西。

我认为，没有适用于每个人的赢得面试的黄金法则。但是，如果你想花时间和精力来练习系统设计面试问题，你最终会发现这些问题非常容易。

如果你有任何问题或建议，请随时给我们发电子邮件，info@gainlo.co。

系统设计面试之前需要知道的八件事

> 原文：Things You Need to Know Before a System Design Interview
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0
>
> 自豪地采用谷歌翻译

面试准备中最常见的问题是什么？ 我可以立即告诉你：如何准备系统设计面试？

很多人害怕系统设计面试，因为没有一定的模式用于准备，问题相当灵活和不可预测。 另外，系统设计的问题通常是开放式的，没有标准或正确的答案，这使得准备过程更加困难。

我们已经花了整整一个月的时间来编写本指南，在系统设计面试之前告诉你一些你应该知道的事情，同时让你更加无忧无虑，因为系统设计面试并不像许多人想象的那样困难，一定的方式绝对可以帮助你适应它。

1. 系统设计面试中评估什么？

我们都知道，编程面试的重点是候选人的基本知识，所以会测试他的一般技术能力和分析能力。

但是，很少有人能清楚地说出系统设计面试的目的。所以在查看提示之前，最好从面试官的角度来理解系统设计面试。

在系统设计面试中，候选人经常被要求设计一个新的系统来解决一个开放式的问题，比如设计短网址服务。有时候这个问题可能相当普遍，比如你如何为 Youtube 设计推荐系统。

在这个过程中，讨论是核心。候选人更有可能主导对话，并和面试官讨论概要组件，细节，优点和缺点，以及一切。

与编程面试相比，系统设计面试更像软件工程师的日常工作。

面试时，主要评估你的沟通和解决问题的能力。给定一个开放的问题，你如何分析这个问题，你如何一步一步解决问题，你如何解释你的想法并与他人讨论，如何评估你的系统并优化它，是面试官最关心的。

所以让我们看看你能做些什么准备。

2. 亲自做项目

准备系统设计面试的最好方法是始终贯穿真实的项目和实践。很多人提前六个月或一年开始准备过程，这绝对是你的最佳实践。

我们看到的一个共同的模式是，你拥有的实践经验越多，在系统设计面试时就越好。这很容易理解，因为这些系统设计问题都来自现实生活中的产品，而那些从事过许多项目的人往往对这些问题有更好的理解，或者这只是他们之前解决的问题之一。

当被要求设计 Youtube 推荐系统时，它与其他许多推荐系统类似，因为很多概念在这里都很常见，

如果你有推荐经验，或者你已经阅读了一些文章/书籍或者思考过，你至少应该能够提出一些初步的想法。

如果你不知道要做什么，这里给你一些建议：

建立一个小型服务/产品来解决一个真正的问题 在 Github 上贡献开源项目 找到你喜欢的机器学习，网络等主题，并搜索一些你可以使用的项目 真正重要的是去做一些现实生活中的项目。你可能需要很长时间才能看到改进，但是在那个时候，你会注意到面试问题是多么的简单。而且，从长远来看，你将从中受益良多。

3. 熟悉基本知识

我不记得我强调了多少次，但这对于系统设计面试非常重要。由于系统设计问题是开放式的，可能涵盖许多技术领域，这里的基本知识远不止是数据结构和算法。

首先，毫无疑问，你应该非常擅长数据结构和算法。以短网址服务为例，如果你不清楚散列，时间/空间复杂度分析，你将无法提出一个好的解决方案。

通常情况下，在时间和内存效率之间有一个权衡，你必须非常精通大 O 分析才能把所有东西都弄清楚。

还有其他一些你最好熟悉的东西，尽管可能不会在面试中涉及。

- 抽象。系统设计面试是一个非常重要的话题。你应该清楚如何抽象系统，什么是可见的和不可见的其他组件，以及它背后的逻辑是什么。面向对象编程也是很重要的。
- 数据库。你应该清楚关于像关系数据库这样的基本概念。取决于你的水平（应届生或经验丰富的工程师），了解 NoSQL 可能是一个加分项。
- 网络。你应该能够清楚地解释，当你在浏览器中键入`gainlo.co`时会发生什么。例如应该清楚 DNS 查找和 HTTP。
- 并发。如果你能够识别系统中的并发问题，并告诉面试官如何解决这个问题，那将是非常棒的。有时候这个话题可能很难，但是需要了解一些基本的概念，比如说竞态条件，死锁是底线。
- 操作系统。有时你与面试官的讨论可能会非常深入，这时最好知道操作系统在底层如何工作。

- 机器学习（可选）。你不需要成为专家，但是一些基本的概念，如特征选择，通常 ML 算法的工作方式，最好熟悉它们。
- 请记住，重点在于要求你从头开始学习所有这些东西，这可能需要一年多的时间。眞正重要的是每个主题背后的基本概念。例如，如果在面试中不能实现神经网络，那么完全没问题，但是你应该可以用一个句子解释它。

4. 自顶向下和模块化

这是解决系统设计问题的一般策略，也是向面试官解释的方法。最糟糕的情况是总是立即深入，只会把事情弄得一团糟。

相反，从概要想法开始，然后一步一步地弄清细节总是不错的，所以这应该是自顶向下的方法。为什么？因为许多系统设计的问题是非常普遍的，没有大方向就无法解决。

以 Youtube 推荐系统为例。我可能会首先把它分成前端和后端（面试官可能只会要求后端或特定部分，但我会涵盖整个系统让你明白）。对于后端，流程可以分为三步：收集用户数据（如他观看的视频，位置，偏好等），离线流水线生成推荐，并存储数据并提供给前端。然后，我们可以深入每个具体组件。

对于用户数据，我们可以列出我们认为的，与用户可能喜欢的视频相关的功能。对于流水线，我们可以讨论如何训练数据集等。我们可以做得更深入。由于Youtube 拥有巨大的数据集，离线流水线必须运行大量的数据，那么可能会使用 MapReduce 或 Hadoop。

我们可以通过逐层深入来继续这种分析，但我想在这里解释的想法是，你应该总是有一个大方向。

另一个技巧是模块化。就像我上面所说的，最好把系统分成不同的组件，这在现实生活中也是很好的做法。模块化不仅可以使你的设计对你和面试官更清晰，而且使测试更容易。

5. 优点和缺点

系统设计问题通常没有太多的限制。 因此，好的解决方案和不好的解决方案之间没有明确的界限。 在这种情况下，你有责任了解不同情况下的最佳方法。

最常见的权衡是在时间和内存之间。 你可以直接告诉面试官每个解决方案的优缺点，并要求他澄清你有多少内存的限制。

另外被要求优化系统的时候，你也可以把一些常见的约束放在那里，例如，如果你正在为驾照设计一些东西，你可以告诉面试官，假设驾照的最大长度可能是七是合理的， 通过这种方式，你可以将所有驾照存储在内存中，从而可以进一步优化系统。

6. 估计

在做估计时，你最好对数字有很好的感觉，这在实际项目中更为重要。更具体地说，你应该清楚地估计出你的系统或程序会消耗多少内存，运行速度有多快，根据你的估计，你将如何调整你的设计。

如果我们使用驾照的示例，那么当然可以假设内存足以存储所有美国驾照。但是，你首先估计你需要多少内存来存储，这会更令人印象深刻。

为了估计内存的消耗，你应该计算最大长度为七时有多少许可证，以及你将使用哪种数据结构来存储，然后计算出你需要多少内存，这会让你弄清楚，你的想法是否是可行的。

当决定存储介质时，内存当然不是唯一的解决方案。除了将所有内容存储在内存之外，你还可以存储在磁盘中，也可以存储在多台计算机上。最佳方法的选择实际上是估计时间和存储成本的问题。找出执行时间和内存限制的瓶颈，将会让你对整个系统有更清晰的认识。

7. 模拟面试

说实话，由于没有标准的答案，所以你自己练习系统设计面试并不容易。所以建议始终在一些有经验的工程师面前练习它。

而且通过这个过程，你会花大部分时间与面试官沟通和讨论，这是系统设计面试的主要内容。所以简而言之，我们强烈建议你不要自己练习系统设计面试。

像 Gainlo 这样的网站可以让你和谷歌，亚马逊等公司的员工进行模拟面试，这真的很有帮助。

8. 从现有系统中学习

这个方法是我通常建议人们去做的。每当你对某个系统感到好奇时，试着弄清楚这个系统如何设计。你可以先尝试自己设计，然后再和实际设计比较。最重要的是，试着理解为什么这样设计。数据大小，速度要求等一些限制可能迫使系统变成这样。

http://highscalability.com 有很多现实世界的系统设计的好文章。对于系统设计面试来说，这可能有些过头了，但是了解它们总是很好的。

你也会注意到，即使是同一种系统，不同的公司也可能有完全不同的设计方式。你肯定会从探索它们中学到很多。

总结

系统设计面试非常有趣，因为它更接近现实世界的产品。 准备的关键在于明确面试中的期望，并花费足够的时间和精力处理真正重要的事情。

如果你对系统设计问题没有标准答案感到害怕，你也可以克服它，因为你的答案始终正确。

我希望这篇文章能够让你不必担心系统设计面试，让我知道你对此有何看法。

如何设计 Twitter（第一部分）

> 原文：How to Design Twitter (Part 1)
>
> 译者：飞龙
>
>
>
> 自豪地采用谷歌翻译

从上周的调查来看，我们的大部分用户都希望更多地了解系统设计面试。

没有任何示例就很难提供提示，这就是为什么在这篇文章中，我将使用一个非常流行的系统设计问题，告诉你如何解决它。

另外值得一提的是，系统设计面试问题可以是非常开放的，因此没有标准答案这样的事情。即使是同一个问题，你也会和不同的面试官进行完全不同的讨论。

因此，不要过分关注问题或解决方案本身。 相反，你最好从分析中总结出某些模式或策略。

我们将从这个“简单”问题开始 - 你将如何设计 Twitter？

注：我从来没有在 Twitter 工作，整篇文章是为了说明系统设计的想法。 另外，感谢 Gainlo 团队在我们一起合作提供所有这些分析。

常见的误解

在我们进行分析之前，我想澄清求职者的最常见的误解之一。

当被要求设计推特时，许多人倾向于立即潜入技术细节。一个常见的行为是列出一些像 MongoDB，Bootstrap，MapReduce 等工具或框架，并试图解释我们应该使用哪种特定的技术。

面试官真正需要的是如何解决问题的概要想法。使用什么工具并不重要，但是如何定义问题，如何设计解决方案以及如何逐步分析问题是非常重要的。

这些正是初级开发人员和高级工程师的区别。如果你还没有查看这个文章，你也应该看看。

定义问题

让我们开始讨论这个问题 - 如何设计 Twitter。

系统设计问题通常是非常普遍的，因此没有很好的定义。这就是很多人不知道如何开始的原因。

这与现实生活中的项目很相似。因此，我们要做的第一件事就是进一步明确问题，使问题更加明确和具体。

在这个问题上，我首先要做的就是将 Twitter 压缩到 MVP（最小可行产品）。换句话说，我们只会设计 Twitter 的核心，而不是一切功能。

所以整个产品应该允许人们相互关注，并查看他人的信息流（Feeds）。就是这样简单。（如果需要任何功能，面试官应该能够澄清）。其它功能例如注册，Moment，安全等都不在讨论范围之内。

概要解决方案

我们之前说过，不要立即深入所有的细节，这也会把面试官和你自己搞晕。

我在这里使用的通用策略是，将整个系统分成几个核心组件。 有相当多的分支策略，例如，你可以分为前端/后端，离线/在线逻辑等。

在这个问题中，我将为以下两件事情设计解决方案：1）数据建模。 2）如何处理信息流。

数据建模 - 如果我们想使用像 MySQL 这样的关系数据库，我们可以定义用户对象和信息流对象。 两种关系也是必要的。 一个是用户可以关注其它用户，另一个是每个信息流都由一个用户拥有。

处理信息流 - 最直接的方法是从所有关注的人中提取信息流，并按时间呈现它们。

面试将不会在这里停止，因为还有很多我们还没有涉及的细节。 面试官完全可以决定接下来要讨论的内容。

具体问题

要记住，概要想法可以无限延伸。所以我只会在这里介绍一些后续问题。

1. 当用户关注很多人时，获取并呈现所有的推文可能开销很大。如何改善它？

   有很多方法。由于 Twitter 拥有无限滚动功能，特别是在移动设备上，每次我们只需要获取最新的 N 个推文，而不是所有的推文。然后会有很多如何实现分页的细节。

   你也可以考虑缓存，这也可能有助于加快速度。

2. 如何检测假用户？

   这可能与机器学习有关。其中一种方法是确定注册日期，粉丝数量，推文数量等相关特征，并构建一个机器学习系统来检测用户是否是假的。

3. 我们可以用其他算法来订阅信息流嘛？

   过去几周这个话题的争论很多。如果我们想根据用户的兴趣订购，如何设计算法？

我想说一些我们应该向面试官澄清的事情。

如何衡量算法？也许用户在推特或用户交互上喜欢/转发的平均时间。

用什么信号来评估用户喜欢这个信息流的可能性？用户与作者的关系，该信息的回复/转发数量，作者的粉丝数量等可能重要。

如果使用机器学习，如何设计整个系统？

4. 如何实现 @ 功能和转推功能？

   对于 @ 功能，我们可以简单存储每个推文的用户 ID 列表。因此，在呈现你的信息流时，你还应该包含 @ 列表中拥有你的 ID 的信息流。这稍微增加了呈现逻辑的复杂性。

   对于转推功能，我们可以做类似的事情。在每个推文中，都会存储一个推文 ID（指针），如果存在，则会指示原始推文。

   但要小心，当用户转推已转推的推文时，你应该能够弄清楚正确的逻辑。这是一个产品决策，你是否要将其制作成多层还是仅保留原始推文。

总结

在这个话题中，我想介绍的东西太多了，所以我不得不把它分成两部分。

这里值得澄清的是，对于我提到的每个问题，没有标准解决方案。 换句话说，还有很多其他解决方案同等或优于上述方案。

另外，这个问题已经被简化了很多，我非常肯定在 Twitter 的生产中已经开发了更多的东西。

在这里再次强调，解决方案不是你应该最关心的。 相反，尝试了解我如何处理和分析问题。

如果你觉得这篇文章有帮助，请分享给你的朋友，我会很感谢。 你也可以在这里查看更多的系统设计面试问题和分析。

如何设计 Twitter（第二部分）

> 原文：How to Design Twitter (Part 2)
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0
>
> 自豪地采用谷歌翻译

这篇文章谈论了系统设计面试的问题，特别是如何设计 twitter 。如果你还没有查看第一部分，请去阅读。

好的，这里是系统设计面试问题 - 如何设计 twitter 的第二部分。上一篇文章中我们已经提到，我们知道系统设计问题可能是非常开放的。即使是同一个问题，你肯定会和不同的面试官有完全不同的讨论。

Gainlo 团队提供了分析，我相当确定有更好的方法。

Twitter 已经是一个拥有大量功能的庞大产品。在这篇文章中，我们将讨论如何从系统设计面试角度设计 Twitter 的特定功能。

热门话题

Twitter 会在主页的搜索页面和左栏（也可能是其他地方）显示热门话题。点击每个主题将你转到所有相关的推文。

问题是如何设计这个功能。

如果你还记得我们在前一篇文章中所说的话，建议先拥有概要方法。简而言之，我会将问题分成两个子问题：1）如何获得热门话题候选？ 2）如何对这些候选进行排名？

对于主题候选，有各种各样的想法。我们可以获得过去的 N 小时内最频繁的标签。我们也可以获得最热门的搜索查询。或者，我们甚至可能会抓取最近最受欢迎的推文，并提取一些常见词语或短语。但个人而言，我会采用前两种方法。

排名可以很有趣。最直接的方法是根据频率进行排名。但是我们可以进一步改进它。例如，我们可以整合像回复/转推/喜欢的数量，新鲜度等信号。我们也可能会添加一些个性化的信号，比如是否有很多关注/粉丝在谈论这个话题。

关注推荐

Twitter 还会向你显示关注推荐。实际上，这是在用户入驻和参与中起着重要作用的核心功能。

如果你玩这个功能，你会注意到 Twitter 主要有两种人给你看 - 你可能认识的人（朋友）和著名的用户（名人/品牌...）。

通过搜索用户的“关注图”，所有这些候选人都不难获得，而三个步之内的人都是很好的候选人。此外，也可以包含大多数追随者的账户。

问题是如何对他们排名，因为每次我们只能显示一些建议。我会倾向于使用机器学习系统来实现它。

有很多我们可以使用的特征，例如其他人是否关注过这个用户，共同关注/粉丝的数量，或任何基本信息（如位置）的重叠等等。

这是一个复杂的问题，有各种后续问题：

- 如何为亿万级用户扩展系统？
- 如何评估系统？
- 如何为 Facebook 设计相同的功能（双向关系）？

Moments

Twitter 会向你展示当前标签中的趋势。这个功能比热门话题更复杂，我认为这里有必要简单地解释一下。

基本上，“Moments”会向你展示不同类别（新闻，体育，娱乐等）的有趣话题列表。对于每个话题，你还可以获得几个讨论它的顶级推文。所以这是一个探索目前情况的很好方式。

我很确定有很多设计这个系统的方法。一种选择是从新闻网站获取过去的 1-2 个小时内最热门的文章。对于每篇文章，找到与其相关的推文，并找出它属于哪个类别（新闻，体育，娱乐等）。然后，我们可以在 Moments 中将这篇文章作为热门话题展示。

另一个类似的方法是获得所有热门话题（与第一部分相同），找出每个话题的类别，在 Moment 中展示。

对于这两种方法，我们将有以下三个子问题来解决：A. 将每个推特/话题分类到一个类别（新闻，体育等）B. 在 Moments 中生成和排列热门话题 C. 为每个话题生成和排名推文。

对于 A，我们可以预先定义几个主题，并进行监督学习。或者我们也可以考虑聚类。事实上，微博中的文字，用户的个人资料，关注者的评论中包含了大量的信息，使算法准确无误。

对于 B 和 C 来说，因为它跟这篇文章的第一部分很相似，所以我就不谈了。

搜索

Twitter 的搜索功能是人们日常使用的另一种流行功能。如果你完全不知道搜索引擎是如何工作的，你可以查看本教程。

如果我们只讨论一般的推文搜索功能（不包括用户搜索和高级搜索），那么概要方法可能与 Google 搜索非常相似，只不过你不需要抓取网页。基本上，你需要建立索引，排名和检索。

如果你深入了解如何设计排名算法，事情会变得非常有趣。与 Google 不同，Twitter 搜索可能更关注新鲜度和社交信号。

最直接的方法是给每个功能/信号一个权重，然后计算每个推文的排名分数。那么我们可以按照分数对它们进行排名。功能可以包括回复/转推/喜欢的数字，相关性，新鲜度，用户的知名度等。

但是，我们如何评估排名和搜索？我认为最好定义一些核心指标，比如每天的搜索总数，搜索之后的推文点击事件等等，每天观察这些指标。他们也是无论怎么改变我们都应该关心的统计量。

总结

你会意识到许多功能都是数据驱动的，这并不意味着你必须是机器学习专家。但是对数据分析如何工作有一些概要性的想法，肯定是有帮助的。

好的工程师通常也会提出很好的问题。每当你有一个解决方案，试着问自己各种问题，来更好地了解系统。

如果你觉得这篇文章有帮助，请分享给你的朋友，我会很感谢。 你也可以在这里查看更多的系统设计面试问题和分析。

创建照片分享应用

> 原文：Create a Photo Sharing App
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0
>
> 自豪地采用谷歌翻译

在我发布我们的第一个系统设计面试问题后 - 如何设计 twitter，我们得到了更多这样的教程的很多请求。

Gainlo 团队亲自挑选了一系列既经典又易于扩展的系统设计面试问题。 因此，我相信这些问题可以帮助你更好地了解如何进行系统设计面试。

同样值得注意的是，Gaino 团队提供的分析仅用于系统设计面试展示。 生产中的真正解决方案可能非常不同，因为我们在这里极大简化了问题。

创建照片分享应用

如何创建像 Instagram 的照片分享应用？

更具体地说，该系统允许人们相互关注，分享/评论/喜欢图片，也许还有一些其他功能，如探索，广告等等。

我们想在这里分析这个问题有几个原因。 首先，图片共享系统非常流行。 我没有选择一个奇怪问题，在现实世界中几乎没有应用。 相反，有很多像 Pinterest，Flickr 等类似的产品。

其次，问题是通用的，这在系统设计面试中是非常普遍的。 通常，面试官不会要求你解决一个明确的问题，这正是让很多人不舒服的东西。

最后，分析涵盖了可扩展性，数据库，数据分析等广泛的主题，以便它可以在其他系统设计面试问题中重用。

概要解决方案

我们在以前的文章中多次强调，建议先从概要解决方案开始，然后再深入研究各种细节。

这种方法的优点是，你能清楚你想要解决什么问题，而且面试官不太可能困惑。

为了设计图片共享系统，确定两个主要对象 - 用户对象和图片对象是相当直接的。

就我个人而言，我想使用关系数据库来解释，因为它通常更容易理解。在这种情况下，我们将有用户表，其中包含名称，电子邮件，注册日期等信息。图片表也是一样。

此外，我们还需要存储两个关系 - 用户关注关系和用户图片关系。这是非常自然的，值得注意的是用户关注不是双向的。

因此，拥有这样的数据模型可以让用户相互关注。为了检查用户的信息流，我们可以从他关注的人中提取所有照片。

潜在的规模问题

上面的解决方案应该可以正常工作。作为一名面试官，我总是想问，当我们拥有数百万用户时，如何解决这个问题会出现什么问题？

这个问题是一个好方法，测试候选人是否可以预见潜在的规模问题，优于只问你如何解决某个问题。

当然，没有标准的答案，我想列举几个想法作为灵感。

1. 响应时间

当用户达到一定数量时，通常会看到响应时间慢成为瓶颈。

例如，一个开销大的操作是呈现用户的信息流。服务器必须检查用户所关注的每个人，从中获取所有图片，并根据特定的算法对其进行排名。当用户关注大量图片的人时，操作可能会很慢。

这里可以应用各种方法。如果是瓶颈，我们可以升级排名算法。如果按照日期排序，我们可以使用无限滚动功能，从每个人中读取前 N 张最近的图片。或者我们可以使用离线流水线，预先计算一些可以加快排名的信号。

关键是，不可能有人关注数百个用户（谁说的？），但可能有人有成千上万的图片。所以核心是加快图片的获取和排名。

2. 架构扩展

当只有数十个用户和图片时，我们可以从一台服务器存储和提供所有的东西。

但是，对于数百万用户来说，单个服务器远远不够，由于存储，内存，CPU 限制等问题。这就是为什么当有大量请求时，很常看到服务器崩溃。

为了扩展架构，经验法则是面向服务的架构（SOA）优于单一的应用。

不要把所有的东西放在一起，最好把整个系统按服务分成小的组件，并把每个组件分开。例如，我们可以将数据库与负载均衡器的 Web 应用（在不同的服务器上）分开。

3. 数据库扩展

即使我们把数据库放在一个单独的服务器上，也不能存储无限的数据。

在某个时候，我们需要扩展数据库。对于这个特定的问题，我们可以通过将数据库分割成像用户数据库，评论数据库等子数据库来进行垂直分割（Partition），或者根据美国用户，欧洲用户等属性进行分割来进行水平分割（Sharding）。

你可以查看这篇文章来深入分析可扩展性问题。

信息流排名

讨论如何在用户的时间线中对信息流（图片）进行排名也很有趣。

尽管按照时间顺序排列所有内容是相当简单的，但这是否是最好的方法？这样的开放式问题在系统设计面试中非常普遍。

其实，可以有不少的选择。 例如，结合时间和用户喜欢这张照片的可能性的算法绝对是有希望的。

为了设计这样的算法，一个常用的策略是提出一个评分机制，将各种特征作为指标并计算每张图片的最终分数。

直观地说，重要的功能包括喜欢/评论的数量，用户是否喜欢许多照片的所有者等等。 由于简单，可以使用线性组合作为起点。

然后，值得尝试像协同过滤这样的更先进的机器学习算法。

图像优化

由于图片共享系统充满了图像，我想问一下可以优化哪些图片相关的内容？

首先，通常建议在生产中分开存储所有照片。 Amazon S3 是最受欢迎的存储系统之一。但是，你不需要能够提出这个。

重点是图像通常很大，很少更新。所以独立的图像存储系统有很多优点。例如，当文件是静态的时候，缓存和复制可以简单得多。

其次，为了节省空间，应该压缩图像。一种常用的方法是仅存储/提供压缩版本的图像。实际上，Google 照片使用这种方式，并提供无限的免费存储。

总结

在这篇文章中，还有一些话题没有涉及，比如如何在 Instagram 中构建探索功能。我希望你可以花一些时间考虑一下。

另外供参考，你可以查看 Instagram 的基础设施和 Flickr 架构。但是，我不认为他们对系统设计面试非常有帮助，因为他们太过专注于技术而不是设计原则。

如果你觉得这篇文章有帮助，请分享给你的朋友，我会很感谢。 你也可以在这里查看更多的系统设计面试问题和分析。

创建短网址系统

> 原文：Create a TinyURL System
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0
>
> 自豪地采用谷歌翻译

如果你已经开始准备系统设计面试，你一定听说过一个最流行的问题 - 创建一个短网址系统。

本周，Gainlo 团队就这个面试问题集思广益。我想总结一下，如何用概要思想来分析这个问题。

如果你是系统设计面试新手，我强烈建议你在系统设计面试之前首先了解一下你需要了解的八件事情，其中提供了可帮助你入门的一般性策略。

问题

如何创建一个短网址系统？

如果你对短网址不熟悉，我在这里简要解释一下。基本上，短网址是 URL 缩短服务，一个 Web 服务，提供简短别名来重定向长 URL 。还有很多其他类似的服务，比如 Google URL Shortener，Bitly 等等。

例如，URL http://blog.gainlo.co/index.php/2015/10/22/8-things-you-need-to-know-before-system-design-interviews/ 很长，很难记住，短网址可以为它创建一个别名 - http://tinyurl.com/j7ve58y。如果你点击别名，它会将你重定向到原始网址。

所以如果你设计这个系统，允许人们输入 URL，生成较短的别名 URL，你会怎么做？

概要思想

让我们以基础和概要解决方案开始，然后再继续优化。

乍一看，每个长 URL 和相应的别名形成一个键值对。 我希望你能马上思考与哈希有关的东西。

因此，问题可以这样简化 - 给定一个 URL，我们如何找到哈希函数 `F` ，它将 URL 映射到一个简短别名：

```
F(URL) = alias
```

并满足以下条件：

每个 URL 只能映射到唯一的别名 每个别名都可以轻松地映射回唯一的 URL

第二个条件是运行其实的核心，系统应该通过别名查找并快速重定向到相应的 URL。

基本解决方案

为了方便起见，我们可以假设别名是`http://tinyurl.com/<alias_hash>`，`alias_hash`是固定长度的字符串。

如果长度为 7，包含`[A-Z, a-z, 0-9]`，则可以提供`62 ^ 7 ~= 3500 亿`个 URL。 据说在写这篇文章的时候，有大约 6.44 亿个网址。

首先，我们将所有的映射存储在一个数据库中。 一个简单的方法是使用`alias_hash`作为每个映射的 ID，可以生成一个长度为 7 的随机字符串。

所以我们可以先存储`<ID，URL>`。 当用户输入一个长 URL `http://www.gainlo.co`时，系统创建一个随机的 7 个字符的字符串，如`abcd123`作为 ID，并插入条目`<"abcd123", "http://www.gainlo.co">`进入数据库。

在运行期间，当有人访问`http://tinyurl.com/abcd123`时，我们通过 ID `abcd123`查找并重定向到相应的 URL `http://www.gainlo.co`。

性能 VS 灵活性

这个问题有很多后续问题。我想在这里进一步讨论的一件事是，通过使用GUID（全局唯一标识符）作为条目 ID，在这个问题中，与自增 ID 相比利弊是什么？

如果你深入了解插入/查询过程，你会注意到使用随机字符串作为 ID 可能会牺牲一点点性能。更具体地说，当你已经有数百万条记录时，插入可能是很昂贵的。由于 ID 不是连续的，因此每次插入新记录时，数据库都需要查看该 ID 的正确页面。但是，使用增量 ID 时，插入可以更容易 - 只需转到最后一页。

所以优化这个的一种方法是使用增量 ID。每次插入一个新的 URL 时，我们都会为新条目增加1。我们还需要一个散列函数，将每个整数 ID 映射到一个 7 个字符的字符串。如果我们把每个字符串看作一个 62 进制的数字，映射应该很容易（当然，还有其他方法）。

另一方面，使用增量 ID 将使得映射更不灵活。例如，如果系统允许用户设置自定义短网址，显然 GUID 解决方案更容易，因为对于任何自定义短网址，我们可以计算相应的哈希作为条目 ID。

注意：在这种情况下，我们可能不使用随机生成的键，而是使用更好的散列函数将任何短网址映射为 ID，例如，一些传统的散列函数，如 CRC32，SHA-1 等。

开销

我很少询问如何评估系统的开销。对于插入/查询，我们已经在上面讨论过了。所以我会更关注存储开销。

每个条目存储为 `<ID，URL>` ，其中 ID 是一个 7 个字符的字符串。假设最大 URL 长度为 2083 个字符，则每个条目需要 `7 * 4 + 2083 * 4 = 8.4 KB` 。如果我们存储一百万个 URL 映射，我们需要大约 8.4G 的存储空间。

如果我们考虑数据库索引的大小，我们也可能存储其他信息，比如用户 ID，日期等等，这肯定需要更多的存储空间。

多台机器

显然，当系统发展到一定规模时，单台机器不能存储所有的映射。我们如何扩展多个实例？

更普遍的问题是如何在多个机器上存储散列映射。如果你知道分布式键值存储，你应该知道这可能是一个非常复杂的问题。我只会在这里讨论概要思想，如果你对所有这些细节感兴趣，我建议你阅读 Dynamo 文章：亚马逊的高可用键值存储。

简而言之，如果要在多个实例中存储大量的键值对，则需要设计查找算法，以便为给定的查找键寻找相应的机器。

例如，如果传入的短名称是 `http://tinyurl.com/abcd123` ，则基于键 `abcd123` ，系统应该知道哪个机器存储了数据库，它包含这个键的条目。这与数据库分片完全一样。

一种常见的方法是让机器充当代理，负责根据查找键，将请求分派到相应的后端存储器。后端存储器是实际上存储映射的数据库。他们可以通过各种方式拆分，如使用 `hash(key) % 1024` 将映射划分到 1024 个存储器中。

有很多细节可以使系统变得复杂，我只是在这里举几个例子：

- 复制。数据存储可能会因各种随机原因而崩溃，因此常见的解决方案是每个数据库都有多个副本。这里可能有很多问题：如何复制实例？如何快速恢复？如何保持读/写一致？
- 重新分片。当系统扩展到另一个级别时，原始分片算法可能无法正常工作。我们可能需要使用新的散列算法来对系统重新分片。如何在保持系统运行的同时，对数据库重新分片，可能是一个极其困难的问题。
- 并发。可以有多个用户同时插入相同的 URL 或编辑相同的别名。用一台机器，你可以用一个锁来控制它。但是，当你扩展到多个实例时，情况会变得更加复杂。

总结

我觉得你已经意识到这个问题乍一看并不复杂，但是当你深入挖掘更多的细节，特别是考虑规模问题的时候，会遇到很多后续问题。

我们在之前的文章中说过，使用的具体技术并不重要。真正重要的是如何解决这个问题的概要思想。这就是我不提 Redis，RebornDB 等东西的原因。

再次，有无限的方式用于进一步扩大这个问题。我想用这篇文章作为起点，来使你熟悉系统设计面试。

如果你觉得这篇文章有帮助，请分享给你的朋友，我会很感谢。 你也可以在这里查看更多的系统设计面试问题和分析。

如何设计 Google Docs

> 原文：How To Design Google Docs
>
> 译者：飞龙
>
>
>
> 自豪地采用谷歌翻译

系统设计面试可能相当开放，需要广泛的知识。

为了准备好这样的面试，重要的是要覆盖不同的领域，而不是集中在单一的主题。我们花了很多时间选择系统设计问题进行分析，我们的主要标准是：

- 这个问题是流行的和经典的
- 我们关心我们选择的问题的多样性
- 分析有助于其他面试问题

本周，我们想讨论如何设计 Google 文档。你会发现它与我们以前的问题的分析有很大的不同。

问题 - 如何设计 Google 文档

我会假设每个人都知道Google文档是什么，不会浪费时间介绍这个产品。

乍一看，这个问题看起来相当普遍，事实确实如此。 Google 文档是一个庞大的系统，具有很多功能。如果你花几分钟时间思考这个问题，则可能会意识到 Google 文档比看起来复杂得多。

作为一个面试官，我不想把讨论的范围限制在这个产品的特定功能上。相反，我倾向于把这个问题提出来，使我能够知道候选人将如何逐步解决一个模糊的问题。

划分为组件

在我们以前的帖子中，我们已经多次强调，当问题很大时，建议提供概要解决方案。抽象解决方案的一种方法是，将大系统分成更小的组件。

显然，Google Docs 是一个庞大的系统，具有一系列功能，包括文档存储，共享，格式化，编辑等。事实上，如果不把它分解成不同的子问题，我几乎不能解决这么大的问题。

如果你没有考虑过这个问题，请在查看我们的分析之前，花费 5-10 分钟的时间自行解答。另外，值得注意的是，如果你的解决方案与我们的解决方案不同，那么问题就是开放的，这是绝对没问题的。

我们可以把整个系统分成以下几个主要部分：

- 文件存储。由于 Google 文档是 Google 云端硬盘的一部分，因此我也包含了存储功能。该系统允许用户将文件（文档）分组到文件夹，并支持编辑/创建/删除等功能。它像一个操作系统。
- 在线编辑和格式化。毫无疑问，Google 文档的核心功能之一就是在线编辑。它支持几乎所有的微软 Office 操作，也许更多。
- 合作。 Google Docs 允许多个人同时编辑单个文档，这眞是太神奇了。这肯定是一个技术挑战。
- 访问控制。你可以与你的朋友分享文档，并给予不同的权限（所有者，只读，允许评论等）。

这里忽略了一些不太重要的功能，像插件，拼写检查，发布到网络等等。

存储和格式化

我将这两个话题放在一起，因为在实现存储和格式化的过程中，会创建一个 Google Docs 非常基本的初级的版本。即使没有访问控制和协作，单个用户仍然可以使用它来编辑文档。

另外，存储和格式化在一定程度上可以看作是后端和前端。

恕我直言，Google Docs（或 Google Drive）的存储系统非常接近操作系统。它有像文件夹，文件，所有者等概念。

因此，构建这样的系统，基本积木是一个文件对象，它包含内容，父项，所有者以及其他元数据，如创建日期。父项代表文件夹关系，根目录的父项为空。我不会讨论如何扩展系统，因为构建分布式系统可能非常复杂。有很多事情要考虑，比如一致性和复制。

对于前端的格式化，一个有趣的问题是，如何使用相应格式储存文档。如果你了解 Markdown，那绝对是最好的解决方案之一。尽管 Google Docs 可能更复杂，但 Markdown 的基本思想仍然适用。

并发

Google Docs 最酷的功能之一是可以同时编辑多个文档。你将如何设计这个功能？

说实话，这不是一个容易的问题。你不能只让每个人自己工作，然后合并每个人的副本或选取最后一次编辑。如果你已经尝试了协作编辑功能，你实际上可以看到对方正在做什么，并获得即时反馈。

如果你已经使用 Git 进行版本控制（译者注：或者简单的 `diff` 和 `patch`），这里的一些想法可以是类似的。首先，让我们考虑最简单的情况 - 只有两个人在编辑同一个文档。假设文档是 `abc`。

基本上，服务器可以为每个人保留量份相同的文档，并跟踪完整的修订历史。当 A 通过在开头添加 `x` 来编辑文档时，这个改变将与 A 所看到的最后修订一起发送到服务器。假设此时 B 删除最后一个字符 `c`，并且这个改变也是这样发送到服务器。

由于服务器知道修改在哪个版本上进行，因此会相应地调整更改。更具体地说，B 的变化是删除第三个字符 `c`，它将被转换为删除第四个字符，因为 A 在开头添加了 `x`。

这就是所谓的操作转换（Operational Tranformation）。如果你从没听说也没关系，基本思想是根据修改和其他合作者的修改来转换每个人的改动。

访问控制

Google 文档允许你使用不同级别的权限邀请合作者。

朴素的解决方案不应该很难。对于每个文件，你可以维护一个合作者列表，带有相应权限如只读，所有者等。当一个人想做特定的行动，系统检查他的权限。

通常，我想问一下这种访问控制系统所面临的挑战是什么。

众所周知，将系统扩展到数百万用户时，可能会有很多问题。我想在这里提到的几件事是：

- 速度。当所有者更新文件夹的权限（例如删除特定的查看者）时，应将此更新传播给其所有子项。速度可能是一个问题。
- 一致性。当有多个副本时，保持每个副本的一致性尤其重要，特别是当多个人同时更新权限时。
- 传播。可能有很多传播情况。除了更新文件夹的权限应该反映在所有的子文件之外，如果你向某人授予文档 D 的读取权限，那么他也可能已经拥有了文档 D 的所有父文件夹的读取权限。如果有人删除了 D 文件，我们可能会撤销 D 的父文件夹的阅读权限（也许不是，这更像是产品决策）。

总结

同样，Gainlo 中没有人从事过 Google Docs。 这篇文章没有教你如何从头开始构建 Google Docs。

相反，我想用这篇文章给你更多想法，有关如何进行系统设计面试和如何解决模糊问题。

设计一个像 Google Docs 这样的复杂系统可能会让人感到恐惧。 但是一旦你把系统分成更小的组件，就变得简单多了。

如果你觉得这篇文章有帮助，请分享给你的朋友，我会很感谢。 你也可以在这里查看更多的系统设计面试问题和分析。

设计新闻推送系统（第一部分）

> 原文：Design News Feed System (Part 1)
>
> 译者：飞龙
>
>
>
> 自豪地采用谷歌翻译

如果你在系统设计面试问题的文章上关注了我们之前的文章，你可能会对新闻推送（译者注：推送、订阅、信息流都是 Feed）系统有多么常见感到惊讶。

无论你是在制作 Twitter，Instagram 还是 Facebook，你都需要某种新闻推送系统来显示来自关注/好友的更新。

实际上，关于新闻推送有一些有趣的细节，比如如何排序，如何优化发布等。所以在这篇文章中，我将介绍这个流行的问题 - 设计新闻推送系统。

问题

为了简单起见，让我们专注于设计 Facebook 的新闻馈送系统，因为不同的产品有不同的要求。

简要总结一下这个功能，当用户进入他们的主页时，他们会以特定的顺序看到好友的更新。推送可以包含图片，视频或文本，用户可能拥有大量的好友。

那么你如何从头开始设计这样的新闻推送系统呢？

子问题

如果你还没有想过这个问题，最好在阅读其余部分之前自行解决。虽然没有标准答案这样的东西，但是通过将你的解决方案与其他人进行比较，仍然可以学到很多东西。

我们开始吧。我们之前说过，面对如此庞大而模糊的系统设计问题时，最好是把大问题分解成子问题，再加上一些概要思想。

对于新闻推送系统，显然我们可以将其分为前端和后端。我将跳过前端，因为这在系统设计面试中并不常见。对于后端来说，三个子问题对我来说似乎至关重要：

- 数据模型。我们需要一些模型来存储用户和推送对象。更重要的是，当我们试图优化系统的读/写时，有很多的权衡。接下来我会详细解释
- 推送排名。 Facebook 的排名不仅仅按照时间顺序排列。

- 推送发布。当只有几百个用户时，发布可能是微不足道的。但是，如果有数百万甚至数十亿的用户，这可能开销较大。所以这里有一个规模问题。

数据模型

有两个基本对象：用户和推文。对于用户对象，我们可以存储用户 ID，名称，注册日期等等。对于推文对象，有推文 ID，推文类型，内容，元数据等，它们也应该支持图片和视频。

如果我们使用关系数据库，我们还需要建立两个关系：用户与推文的关系和好友关系。前者非常简单。我们可以创建一个用户-推文表，存储用户 ID 和相应的推文 ID。对于单个用户，如果他已经发布了多个推文，则可以包含多个条目。

对于好友关系，邻接表是最常见的方法之一。如果我们将所有的用户看作巨型图中的节点，连接节点的边表示好友关系。我们可以使用好友表来为边（好友关系）建模，它的每个条目包含两个用户 ID。通过这样做，大部分的操作都非常方便，例如获取用户的所有朋友一样，检查两个人是否是好友。

数据模型 - 续

在上面的设计中，让我们看看，当我们从用户的所有朋友获取推送时，会发生什么。

系统将首先从好友表中获取所有好友的用户 ID。然后从推文表中为每个好友提取所有的推文 ID。最后，根据推文表中的推文 ID 提取推文内容。你可以看到我们需要执行三个连接，这会影响性能。

一个常见的优化是将推文内容和推文 ID 一起存储在用户-推文中，这样我们就不需要再连接推文了。这种方法被称为去规范化，这意味着通过添加冗余数据，我们可以优化读取性能（减少连接数量）。

> 译者注：这里有些问题。由于每个推文只有一个作者，所以作者 ID 完全可以放进推文表里面，而且没有冗余。

缺点是显而易见的：

- 数据冗余。我们正在存储冗余数据，占用存储空间（经典的时空关系）。
- 数据一致性。每当我们更新推文时，我们需要更新推文表和用户-推文表。否则，数据不一致。这增加了系统的复杂性。

请记住，没有一种方法总是比其他方法更好（规范化与去规范化）。这是你想要优化读取还是写入的问题。

排名

排列推文最直接的方法是创建时间。显然，Facebook 做的不止于此。“重要”的推文排在最前面。

在跳到排名算法之前，我通常会问为什么要改变排名？我们如何评估新排名算法是否更好？如果候选人自己提出这些问题，这绝对令人印象深刻。

优化排名的原因不是看起来是对的。相反，一切都应事出有因。假设有几个我们关心的核心指标，例如用户粘性，留存率，广告收入等。更好的排名系统可以显着改善这些指标，这也回答了如何评估是否取得进展。

所以回到问题 - 我们应该如何排列推文？一个常见的策略是根据各种特征计算推文得分，并根据分数对推文进行排名，这是所有排名问题最常用的方法之一。

更具体地说，我们可以选择几个与推文重要性最相关的特征，例如，分享/喜欢/评论的数量，更新的时间，推文是否有图像/视频等。然后，可以通过这些特征来计算得分，可能是线性组合。对于一个简单的排名系统来说，这通常足够了。

总结

在写这篇文章之前，我没有想到会有这么多细节，所以我不得不把这个文章一分为二。

在第二部分中，我们将讨论排名，推文发布的扩展性问题，以及其他有趣的主题的更多细节。

如果你觉得这篇文章有帮助，请分享给你的朋友，我会很感谢。 你也可以在这里查看更多的系统设计面试问题和分析。

设计新闻推送系统（第二部分）

> 原文：Design News Feed System (Part 2)
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0
>
> 自豪地采用谷歌翻译

这是系统设计面试问题分析 - 设计新闻推送系统的第二部分。如果你还没有看我们的第一篇文章，请查看它。

这里只是简要地总结一下我们在第一部分中讨论过的内容。我们从一个简单的问题开始 - 如何设计 Facebook 的新闻推送系统，允许用户看到来自好友的推送/更新。我们使用关系数据库对整个系统进行建模，并讨论了不同设计的优缺点。

排名是新闻推送系统的一个有趣的话题。我们在前一篇文章中解释了一些整体的排名思路。在这篇文章中，我们将继续讨论排名，还包括推送发布等主题。

排名 - 续

排名的整体思路是首先选择相关的特征/信号，然后弄清楚如何组合它们来计算最终分数。这种方法在很多现实系统中非常普遍。

正如你所看到的，这里重要的是两件事情 - 特征和计算算法。为了让你更好地了解它，我想简单介绍一下 Facebook 上的排名实际上的工作方式 - EdgeRank。

对于每个新闻更新，当其他用户与该推文交互时，他们正就创建了一个东西，Facebook 称之为“边”，其中包括例如喜欢和评论之类的操作。

首先，我们来看看使用哪些特性来评估更新/推送的重要性。边的排名基本上是使用三个信号：亲密度，边权重和时间衰减。

- 亲密度（`u`）。对于每个新闻推送源，亲密度都会评估你与此用户的距离。例如，你更可能关心你的亲密好友的推送，而不是刚刚遇到的人。你可能会问如何计算亲密度，我很快会讨论它。
- 边权重（`e`）。边权重基本上反映了每个边的重要性。例如，评论比喜欢更重要。
- 时间衰减（`d`）。故事越老，用户越不喜欢它。

那么 Facebook 如何通过这三个特征进行排名呢？计算算法非常简单。对于你创建的每个推文，将每条边的这些因子乘起来，然后将边的得分相加，就得到了更新的 EdgeRank。而且它越高，你的更新就越有可能出现在用户的推送中。

亲密度

我们可以做同样的事情来评估亲密度。

可以用各种因素来反映两个人的距离。首先，像评论，标签，分享，点击等显式交互是我们应该使用的强有力的信号。显然，每种类型的互动应该有不同的权重。例如，评论应该比喜欢多得多。

其次，还要跟踪时间因素。也许你曾经和一个好友进行过很多的交流，但最近还是比较少见。在这种情况下，我们应该降低亲密度。所以对于每一个互动，我们也应该考虑时间衰减因子。

作为排名部分的总结，我希望这个排名的常用方法可以成为你的收获。另外，EdgeRank 在 2010 年首次发布，可能已经过时了。

推送发布

当用户加载来自他的好友的所有推送时，这可能是开销很大的行为。请记住，用户可能有成千上万的好友，他们每个人都可以发布大量的更新，特别是对于高端用户。要加载来自好友的所有推文，系统至少需要两个连接（获取好友列表和推文列表）。

那么如何优化和扩展推送发布系统呢？

基本上有两种常见的方式 - 推和拉。

对于推送方式，一旦用户发布了一个推文，我们立即将这个推文（实际上是推文的指针）推送给他的所有好友。优点是在提取推文时，不需要浏览好友列表并获取来自每个好友的推文。它显着减少了读取操作。但是，缺点也是显而易见的。它增加了写操作，特别是对于有大量好友的人来说。

对于拉取方式，只有用户正在加载其主页时，才会提取推文。因此，推文数据在创建后不需要立即发送。你可以看到这种方法优化了写操作，但是即使在使用非规范化之后，获取数据也可能很慢（如果你不了解它，请查看我们之前的文章）。

这两种方法在某些情况下都能很好地工作，而且最好了解它们的优点和缺点。

选择性扇出（fanout）

将活动推给所有好友或粉丝的过程称为扇出。所以推送方式也被称为写时扇出，而拉入方式则是加载时扇出。

在这里，我想问问你是否有任何方法，来进一步优化扇出过程？

事实上，你可以将两者结合。具体而言，如果你主要使用推送方式，则可以做的是，禁用高级用户的扇出，而其他人只能在读取期间加载更新。这个想法是，推送操作对于高级用户来说可能开销很大，因为他们要通知很多好友。通过禁用他们的扇出，我们可以节省大量的资源。实际上 Twitter 采用这种方法后已经有了很大的改进。

同样的道理，一旦用户发布一个推文，我们也可以将扇出限制在他的活跃好友。对于非活跃用户，大多数时候推送操作是个浪费，因为他们将永远不会回来使用推送。

总结

如果遵循 80-20 规则，则 80% 的成本来自 20% 的功能/用户。因此，优化确实涉及瓶颈的确定。

此外，推送系统是一个非常受欢迎的话题，因为它现在被如此多的产品广泛使用。如果你对这个主题感兴趣，并想进一步探索，我建议你看看下面的资源：

- 雅虎研究的新模型
- Etsy 的活动推送架构
- Twitter 的方法

设计 Facebook 聊天功能

> 原文：Design Facebook Chat Function
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0
>
> 自豪地采用谷歌翻译

准备系统设计面试的最有趣的部分之一，就是你可以了解很多如何构建现有系统的细节。

为了使每周的博文更有帮助，我想涵盖广泛的主题。我们一直在谈论推荐这样的东西，在过去的几周里排名很高，这次我想要介绍一些不同的东西。

问题

我们从一个非常简单的问题开始 - 如何设计 Facebook 聊天功能？

Facebook 花 $19B 购买了 Whatsapp，而 Facebook Messenger 最近真的很受欢迎，聊天功能绝对是一个热门话题。所以在这篇文章中，我很乐意谈论它。

这里提到几件事情。首先，正如我在之前的文章中提到的，系统设计面试可能非常多样化。这主要取决于面试官决定讨论哪个方向。因此，即使在相同的问题上，不同的面试官也可能有完全不同的讨论，你不应该期望这篇文章是一个标准答案。

另外，我从来没有在 Facebook Messenger 或 Whatsapp 中工作过。所有的讨论都基于 Gainlo 团队的分析。

基础设施

如前所述，最好有一个概要解决方案，并谈论整体的基础设施。如果你之前没有消息应用的经验，你可能会发现想出一个基本的解决方案并不容易。但是这完全没问题。让我们想出一个非常朴素的解决方案，并在稍后进行优化。

基本上，构建消息应用最常用的方法之一，就是拥有一个聊天服务器，作为整个系统的核心。消息到达时，不会直接发送给接收者。相反，它会转给聊天服务器并先存储在那里。然后，根据接收者的状态，服务器可以立即向他发送消息或发送推送通知。

更详细的流程如下所示：

- 用户 A 想要向用户 B 发送消息 `Hello Gainlo` 。首先将消息发送到聊天服务器。
- 聊天服务器收到该消息，并将确认发送回 A，表示收到该消息。根据该产品，前端可能

会在 A 的用户界面中显示单复选标记。

- 情况1：如果 B 在线并连接到聊天服务器，那很好。聊天服务器只是将消息发送给 B.
- 情况2：如果 B 不在线，则聊天服务器向 B 发送推送通知。
- B 收到该消息，并向聊天服务器发回确认。
- 聊天服务器通知 A，B 接收到消息，并在 A 的用户界面中使用双复选标记进行更新。

实时

一旦达到一定的水平，整个系统可能开销大而低效。那么我们可以通过什么方式来优化系统，以便支持大量的并发请求？

有很多方法。这里一个显而易见的开销就是，当向接收者传递消息时，聊天服务器可能需要产生 OS 进程/线程，最后初始化 HTTP（可能是其他协议）请求并关闭连接。事实上，每个消息都会发生这种情况。即使我们采取相反的方式，接收者一直请求服务器来检查是否有新消息，但仍然开销很大。

一个解决方案是使用 HTTP 长连接。简而言之，接收者可以通过长连接发出 HTTP GET 请求，直到聊天服务器返回任何数据才会返回。每个请求将在超时或中断时重新建立。这种方法在响应时间，吞吐量和开销方面有很多优势。

如果你想了解 HTTP 长连接的更多信息，你可以查看 BOSH。

在线通知

Facebook 聊天的另一个很酷的功能是显示在线的朋友。虽然这个功能乍一看似乎很简单，但它极大地改善了用户体验，绝对值得讨论。如果你被要求设计这个功能，你会怎么做？

显然，最简单的方法是，一旦用户在线，他会向所有的朋友发送通知。但是，如何评估这个开销？

在高峰时段，我们大致需要 `O(平均朋友数*峰值用户数)` 的请求，这在有数百万用户的情况下可能会很多。而这个开销甚至可能比信息开销本身更高。改进它的一个想法是，减少不必要的请求。例如，只有当这个用户重新加载页面或发送消息时，我们才发出通知。换句话说，我们可以将范围限制为“非常活跃的用户”。或者，用户已经在线五分钟之前，我们不会发送通知。这解决了用户显示在线并立即离开的情况。

总结

还有很多其他的话题，我都没有在这篇文章中讨论，例如，如果你深入了解网络，我们可以讨论在连接中可以使用什么网络协议。另外，如何处理系统错误和复制数据也可以是有趣的，因为聊天应用是完全不同的。

如果你想和我进一步讨论，请随时留下评论。

如何为 Twitter 设计趋势算法

> 原文：How to Design a Trending Algorithm for Twitter
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0
>
> 自豪地采用谷歌翻译

从 Twitter 的起初，趋势话题已经成为这个流行产品的核心特征之一。从 Twitter 的趋势中，你可以很容易得知现在流行什么。难怪许多公司喜欢在系统设计面试中让候选人设计趋势算法。

我们在之前的文章 - 如何设计 Twitter（第二部分）中简要介绍了这个主题。由于这个问题如此受欢迎，我决定写一篇它的深度文章。

同样，Gainlo 没有一个人在 Twitter 工作过，不同的面试官也有自己的系统设计面试风格。所以，这篇文章的重点绝对不是给你这个问题的标准答案，而是给你这个话题的更多想法。

问题

由于 Twitter 如此流行，我只会在这里简单地澄清这个问题。

对于为 Twitter 设计一个趋势算法，系统应该能够提供当前流行的主题列表。我喜欢把这个问题弄得一般而且有点含糊，因为我想看看候选人如何处理这样一个开放式的问题。例如，这里的流行没有明确定义。

一般想法

如果你之前没有想过这个问题，我会建议你至少花 15 分钟考虑一下。有自己的解决方案，然后与其他人比较总是比较好。

好的，我们开始。

一个普遍的想法是让我们用一个词（或一个词）来表示一个主题，它可以是一个标签（如 `#gainlo`）或者只是一个单词（如唐纳德·特朗普）。如果与过去相比，一个术语在最近的推文中大量提到，那么这个术语应该被认为是流行的。例如，如果今天有数百万人在讨论 `#gainlo`，但是过去只有数百人谈论这个问题，`#gainlo` 当然应该是当前的热门话题。

我们应该与过去的检索量比较，是因为对于“星期一”或“天气”这样的常见词汇来说，它们的检索量在任何时候都非常大，在大多数情况下都不应该选择为趋势。

综上所述，基本思路是，对于每个术语而言，如果最近几个小时内的检索量与前 X 天的检索量之比很高，则将其视为一个热门话题。

基础设施

当然，热门话题应该立即显示，这意味着我们可以要求用户等待一个小时，以便系统可以计算和排列所有术语。 那么最低的基础设施是什么样的？

显然，考虑到每天的大量推文，计算可能开销较大。 在这种情况下，我们可以考虑使用离线流水线。

更具体地说，我们可以让多个流水线在离线状态下运行，计算每个术语的比率并将结果输出到某个存储系统。 假设短时间内没有大的差别，流水线可以每隔几小时刷新一次。 所以当用户从前端检查热门话题的时候，我们可以预先计算出这个结果。

基本的做法真的很朴素，你有什么想法来改进它吗？ 可以从任何角度来看，如降低系统开销，提高数据质量等。下面我将介绍几点思路。

绝对检索量

如果你只是按照上面的解释计算比例，我很肯定会选择一些非常奇怪的术语。想想下面的情况，假设只有 300 人在推特上发表一个奇怪的话题 `#gailo-mock-interview` ，过去从来没有人谈论过这个话题。比率（过去几小时内的检索量/过去 X 天内的检索量）为 1，可以列在列表的顶部。

显然，这不是我们想要展示给用户的东西。我相信你已经在这里发现了问题。如果绝对量不够大，可能会选择一些不受欢迎的术语。你可以计算 `最近几个小时内的检索量/(最近 X 天内的检索量+ 10000)` 的比值，这样小的检索量就会被稀释。或者，你可以使用单独的信号作为绝对检索量得分与比值结合使用。

意见领袖

另一个想法是，如果一些话题由高端人士讨论，他们可能更有趣，更受欢迎。

有很多方法来设计这个算法。一种方法是首先确定谁是高端人士。我们可以简单地使用粉丝数（虽然有很多冒牌的意见领袖买了粉丝）。如果某个主题被任何意见领袖发推，我们可以计算这个主题推了多少次。因此，只需基于意见领袖的流行度将推文数量乘以一个参数即可。

有人可能会争辩说，我们不应该给意见领袖更多的权重，因为如果一个话题是趋势，就必须有大量的普通用户在谈论这个话题。这可能是真的。这里的重点是，直到你试一试，你才会知道结果。所以我绝对不是说这是正确的做法，但可能值得做一个实验。

个性化

不同的人有不同的品味和兴趣。我们可以根据不同的用户调整趋势列表。这可能是相当复杂的，因为你可以做很多事情来使其个性化。

例如，你可以根据一些信号，包括他以前的推特，他跟随的人和他喜欢的推文等，计算每个主题和用户之间的相关性分数。然后可以将相关性分数与趋势比率一起使用。

另外，位置也应该是一个有价值的信号。我们甚至可以计算每个地点（也许是城市级别）的趋势主题。

总结

还有更多有趣的话题我还没有涉及。例如，很多话题都是一样的，系统如何对它们去重复呢？有没有其他信号可以被整合，比如用户搜索历史？

趋势算法是非常有趣和有用的。尽管我们使用 Twitter 作为例子，但我们讨论过的很多东西也可以应用到 Facebook 或其他平台上。

设计缓存系统

> 原文：Design a Cache System
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0
>
> 自豪地采用谷歌翻译

与我们以前的文章相似，我们希望选择一些流行和实用的系统设计面试问题，这样你不仅可以理解如何在面试中分析问题，同时还可以学习一些有趣的内容。

如果你不了解系统设计面试，我建议你先阅读本教程。在这篇文章中，我们正在解决这个问题 - 如何设计一个缓存系统。这篇文章涵盖的主题包括：

- LRU 缓存
- 换页策略
- 缓存并发
- 分布式缓存系统

问题

如何设计缓存系统？

缓存系统是目前几乎所有应用中广泛采用的技术。另外，它适用于技术栈的每一层。例如，在网络领域中，缓存用于 DNS 查找，Web 服务器缓存用于频繁的请求。

简而言之，缓存系统（可能在内存中）存储常用资源，当下次有人请求相同的资源时，系统可以立即返回。它通过消耗更多的存储空间来提高系统效率。

LRU

最常用的缓存系统之一是 LRU（最久未使用）。 事实上，另一个常见的面试问题是讨论 LRU 缓存的数据结构和设计。 我们从这个方法开始。

LRU 缓存的工作方式非常简单。 当客户端请求资源 A 时，发生如下情况：

- 如果缓存中存在 A，我们只需立即返回。
- 如果没有，并且缓存具有额外的存储空间，则我们获取资源 A 并返回给客户端。 另外，将 A 插入缓存。
- 如果缓存已满，我们将最久没使用的资源剔除，并将其替换为资源 A

这里的策略是最大限度地提高请求资源存在于缓存中的机会。 那么我们怎样才能实现一个简单的 LRU？

LRU 设计

LRU 缓存应该支持这些操作：查找，插入和删除。显然，为了实现快速查找，我们需要使用散列。同样的道理，如果我们想要快速插入/删除，链接列表就会出现在你的脑海里。由于我们需要有效地查找最久未使用的项目，所以我们需要按顺序排列队列，栈或有序数组。

为了结合所有这些分析，我们可以使用由双向链表实现的队列来存储所有的资源。此外，还需要一个哈希表，其中资源标识符为键，相应队列节点的地址为值。

工作原理是这样。当请求资源 A 时，我们检查哈希表来查看缓存中是否存在 A。如果存在，我们可以立即找到相应的队列节点并返回资源（译者注：并移动到队列尾部）。如果不是的话，我们将 A 添加到缓存中。如果有足够的空间，我们只要在队列的末尾添加 A 就可以了。否则，我们需要删除最久未使用的条目。为了这样做，我们可以很容易地删除队列的头部和哈希表中相应的条目。

（译者注：队列中也需要储存资源 ID，这样从队列中删除资源之后，可以从哈希表中也删除。）

换页策略

当缓存已满时，我们需要删除现有项目来存放新资源。 实际上，删除最久未使用的项目只是最常用的方法之一。 那么还有其他方法可以做到吗？

如上所述，策略是尽可能地将请求资源存在于缓存中。 我将在这里简要提一下几种方法：

- 随机替换（RR） - 如术语所示，我们可以随机删除一个条目。
- 最不经常使用（LFU） - 我们维护每个项目的请求频率，并删除最不经常使用的项目。
- W-TinyLFU - 我也想谈谈这个现代的换页策略。 总而言之，LFU 的问题在于，有时候一个条目只是过去经常使用，而 LFU 仍然会保留这个项目很长一段时间。 W-TinyLFU 通过计算时间窗内的频率来解决这个问题。 它也有各种存储优化。

并发

为了讨论并发性，我想谈谈为什么缓存存在并发问题，我们如何解决这个问题。

它可以归为经典的读写器问题。当多个客户端同时尝试更新缓存时，可能会有冲突。例如，两个客户端可能竞争相同的缓存槽，而最后一个更新缓存的客户端将获胜。

当然，常见的解决方案是使用锁。缺点是显而易见的 - 它会严重影响性能。我们如何优化呢？

一种方法是将缓存分成多个分片，并为每个分片分配一个锁，这样如果客户端在不同的分片中更新缓存，就不会相互等待。但是，由于热门的条目更有可能被访问，某些分片将比其他碎片锁定得更频繁。

另一种方法是使用提交日志。为了更新缓存，我们可以将所有改动存储到日志中，而不是立即更新。然后一些后台进程将异步执行所有的日志。数据库设计中通常采用这种策略。

分布式缓存

当系统达到一定规模时，我们需要将缓存分配给多台机器。

一般的策略是保留一个哈希表，将每个资源映射为相应的机器。 因此，当请求资源 A 时，从这个哈希表中我们知道机器 M 负责缓存 A 并将请求指向 M。在机器 M 中，其工作方式类似于上面讨论的本地缓存。 如果 A 不存在于内存中，则机器 M 可能需要获取并更新 A 的缓存。 之后，它将缓存返回到原始服务器。

如果你对此主题感兴趣，可以查看更多 Memcached 的信息。

总结

缓存可能是非常有趣和实用的话题，因为现在几乎所有的系统都使用它。 还有很多话题，像过期策略，我没有在这里设计。

如果你想了解更多类似的文章，请查看我们的系统设计面试问题集。

设计推荐系统

> 原文：Design a Recommendation System
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0
>
> 自豪地采用谷歌翻译

我们从一个简单的问题开始：如何设计推荐系统？

似乎这个问题在最近的系统设计面试中被多次闻到。另外，现在推荐系统非常重要，几乎每个公司都有自己的推荐系统，可以用来提供各种建议。所以这个话题讨论起来可能相当有趣。

在这篇文章中，我将主要讨论推荐系统的各个方面，可能在系统设计面试中讨论。将涉及的主题包括推荐系统，协同过滤（CF），推荐系统基础设施等。

推荐系统

推荐系统一直是一个热门话题。似乎几乎每个公司都在建立这样的系统。例如，亚马逊正在使用推荐系统来提供客户可能也喜欢的商品。 Hulu 正在使用推荐系统来推荐其他受欢迎的节目或剧集。

为了限制讨论范围，我们主要关注 Youtube 的推荐系统。更具体地说，系统负责推荐用户可能喜欢观看的视频。

启发式解决方案

虽然机器学习（ML）通常用于建立推荐系统，但并不意味着它是唯一的解决方案。有很多情况下，我们想要更简单的方法，例如，我们可能有很少的数据，或者我们可能想快速建立一个最小的解决方案。

在这种情况下，我们可以从一些启发式解决方案开始。事实上，我们可以实现很多黑魔法，来构建简单的推荐系统。例如，根据用户观看的视频，我们可以简单推荐同一作者的视频。我们也可以推荐标题或标签类似的视频。如果我们使用知名度（评论数量，分享数量）作为另一个信号，则这个推荐系统作为一个底线，可以运行得很好。

协同过滤

在谈到推荐系统时，我很难避免提到协同过滤（CF），这是推荐系统中最流行的技术。由于不是每个人都有机器学习的背景，所以我不会更深入地介绍这个算法。实际上，协同过滤的美妙之处在于基本思想非常简单，每个人都可以很容易的理解它。

简而言之，为了向用户推荐视频，我可以提供类似用户喜欢的视频。例如，如果用户 A 和 B 已经观看了一堆相同的视频，则用户 A 很可能喜欢 B 所喜欢的视频。当然，在这里定义什么是“相似”有很多方式。这可能是两个用户喜欢同一个视频，也可能意味着他们拥有相同的位置。

以上算法被称为基于用户的协同过滤。另一个版本称为基于条目的协作过滤，意思是推荐的视频与用户观看过的视频类似（条目）。

特征工程

事实上，在系统设计面试中提到协同过滤并不令人印象深刻，因为算法非常普遍。大多数面试官关心的是如何构建针对面试问题的系统。那么对于 Youtube 视频推荐，可以使用哪些功能来构建推荐系统？

通常有两种类型的特征 - 显式和隐式特征。显式特征可以是收视率，收藏夹等。在 Youtube 上，它可以是喜欢/共享/订阅行为。隐式特征不太明显。如果用户只观看了几秒钟的视频，可能是一个负面的迹象。给定一个推荐视频的列表，如果用户点击一个而不是另一个，这可能意味着他喜欢点击那个。通常，我们需要深入探讨隐式特征。

回到Youtube的问题，有几个特征是相当明显的：

- 喜欢/分享/订阅 - 如上所述，它们是关于用户喜好的强烈信号。
- 观看时间
- 视频标题/标签/类别
- 新鲜度

值得大家注意的是，构建机器学习系统的时候，你必须尝试大量的不同特征组合。如果不尝试，你就不会知道哪一个是好的。

基础设施

推荐系统是很好的系统设计面试问题，另一个原因是它也可以用来讨论基础设施。显然，系统包含多个步骤/组件。那么你如何设计整个系统的基础设施呢？

鉴于在 Youtube 上对比类似的用户/视频可能是耗时的，这部分应该在离线流水线中完成。因此，我们可以把整个系统分为在线和离线。

对于离线部分，所有的用户模型和视频需要存储在分布式系统中（有一整篇文章都在讲存储，本文简要介绍了这个话题）。计算相似用户/视频的流水线也会定期运行，以便保持数据更新。事实上，对于大多数机器学习系统来说，使用离线流水线来处理大数据是很常见的，因为你不会期望它在几秒钟内完成。

对于在线部分，根据用户个人资料和他的行为（如刚刚观看的视频），我们应该能够提供来自离线数据的推荐视频列表。通常情况下，系统会获取比所需更多的视频，然后进行过滤和排序。我们可以过滤与用户观看过的视频无关的视频。然后我们也应该排列这些建议。一些因素应该考虑进来，包括视频流行度（分享/评论/喜欢的数量），新鲜度，质量等方面。

总结

实际上，还有很多方法可以用于改进我们尚未涉及的系统。 我想简单地提一些技巧：

新鲜度可能是一个非常重要的因素。 我们应该弄清楚如何推荐新鲜的内容。 评估是推荐系统的重要组成部分，它使我们能够了解系统的工作情况。 为了训练协作过滤系统，我们还可以包括视频位置信号。 通常情况下，排名靠前的视频有更高的点击机会。 很难预测系统设计面试中会讨论什么，这就是我尽量在文章中涵盖尽可能多的话题，而不是深入到特定的领域的原因。

如果你觉得这篇文章有帮助，请分享给你的朋友，我会很感谢。 你也可以在这里查看更多的系统设计面试问题和分析。

随机 ID 生成器

> 原文：Random ID Generator
>
> 译者：飞龙
>
>
>
> 自豪地采用谷歌翻译

让我们继续讨论我们的系统设计面试问题。如果你刚刚看到这个系列，你可以查看我们以前的文章。基本上，我们每个星期都会选择一些有趣的面试问题，并提供深入的分析。

值得注意的是，这篇文章并不是要给你一些标准答案。相反，我们更注重分析问题，以及如何提出合理的方法。系统设计面试更是如此，因为这个问题可能是非常开放的。

本周，我们将讨论如何设计随机 ID 生成器。我们将介绍一些主题，包括扩展 ID 生成器和每种方法的优缺点。

随机 ID 生成器

如果你不知道ID生成器是什么，请让我在这里简要解释一下。假设你正在构建 Twitter 这样的社交网络产品，则需要将每个用户存储在数据库中，并使用唯一的用户标识来确定系统中的每个用户。

在某些系统中，你可以从 `1, 2, 3 ... N` 持续增加 ID 。在其他系统中，我们可能需要生成一个随机字符串的 ID。通常，ID 生成器的要求很少：

他们不能任意长。比方说，我们保持在 64 位。 ID 按日期递增。这给系统提供了很大的灵活性，例如你可以通过 ID 排序用户，这与登记日期排序相同。

还有一些其他的要求，尤其是当你想扩展系统来支持数百万甚至数十亿用户时。

单机

我们以前的建议是，从简单的事情开始并继续优化就很好，当问题很宽泛时更是如此。如果我在系统设计面试中遇到这个问题，很可能我会从一台机器设计开始。这不仅容易设计，而且在大多数情况下都足够了。

在最简单的情况下，我们可以从 `1, 2, 3 ... N` 递增 ID，这实际上是许多真实项目中最常用的生成 ID 的方法之一。如果用户 A 的 ID 比用户 B 大，那么我们知道 A 是稍后注册的。

但是，这种方法难以扩展。比方说一年之后，每天都有太多的用户，需要将数据库扩展到多个实例。你会看到这种方法无效，因为它可能会为不同的用户生成重复的 ID。

第三方服务

为了将 ID 生成器扩展到多台机器，一种自然的解决方案是维护一个单独的服务器，只负责生成 ID。更具体地说，当用户注册产品时，无论哪个服务器处理这个请求，它都会连接到第三方服务器来请求一个随机 ID。由于所有 ID 的生成都是在一台服务器上处理的，因此不会产生重复的 ID。

然而，这个解决方案的缺点是显而易见的。假设产品如此受欢迎，以至于一秒钟内就可能有大量的注册用户，第三方服务器很快就会成为瓶颈。服务器可能会阻止注册或只是崩溃。

多机解决方案

因此，我们必须将 ID 生成扩展到多个服务器。如果我们不想和 ID 生成服务器通信，则每个服务器应该自己能够生成随时间增加的唯一 ID。考虑使用时间戳来生成 ID 应该是很自然的。

由于在一个时间戳内也可能有多个用户，我们可以用两种方法解决这个问题。

- 我们为每个 ID 生成服务器分配一个服务器 ID，最终的 ID 是时间戳和服务器 ID 的组合。
- 我们还可以在单个服务器上的单个时间戳内允许多个请求。我们可以在每个服务器上保留一个计数器，它表示在当前时间戳里已经生成了多少个 ID。所以最终的 ID 是时间戳，serverID 和计数器的组合。

如前所述，ID 不能是任意长的，例如计数器可能只有 8 位。在这种情况下，服务器最多可以在单个时间戳内处理 256 个请求。如果频繁超过这个限制，我们需要添加更多的实例。

事实上，这个解决方案就是 Twitter 解决问题的方法。他们开放了他们的 ID 生成器，叫 Snowflake。

时钟同步

我们忽略了上述分析中的一个关键问题。 事实上，有一个隐藏的假设，即所有 ID 生成服务器都有相同的时钟来生成时间戳，在分布式系统中可能不是这样。

实际上，在分布式系统中的系统时钟可能会发生严重偏斜，这可能会导致我们的 ID 生成器提供重复的 ID，或顺序不正确的 ID。 时钟同步不在本次讨论的范围内，但是，了解这个系统中的这个问题对你来说很重要。 有很多方法可以解决这个问题，如果你想了解更多的信息，请查看 NTP。

总结

随机 ID 生成器在许多真实项目中是一个非常实际的问题。 与许多其他系统类似，乍看起来似乎很容易，但在扩展到多台机器时存在相当多的问题。 如果你想了解此主题的更多信息，还可以查看 Flickr 的票务服务器，这是除了“Snowflake”之外的另一种方法。

设计键值存储（第一部分）

> 原文：Design a Key-Value Store (Part I)
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0
>
> 自豪地采用谷歌翻译

由于很多人给我们发邮件说，他们想了解系统设计面试的更多内容，我们将会介绍更多这个话题。 我很高兴听到很多反馈意见，如果你有任何建议或疑问，请发表评论告诉我们。

本周，我要谈谈键值存储。 键值存储是世界上几乎所有系统都使用的非常强大的技术。 它可以像散列表一样简单，同时也可以是分布式存储系统。 例如，Cassandra 的下划线系统是一个键值存储系统，Cassandra 在苹果，Facebook 等许多公司中被广泛使用。

在这篇文章中，我将介绍基本的键值存储系统，分布式键值存储以及包括分片在内的扩展问题等，这些都可能在系统设计面试中涵盖。

基本的键值存储

如何在一台机器上设计简单的键值存储系统？

最直接的方法是使用散列表来存储键值对，这是目前大多数这类系统的工作原理。散列表允许你在常数时间内读/写一个键值对，而且使用起来非常简单。大多数语言都有内置的支持。

但是，缺点也是显而易见的。使用哈希表通常意味着你需要将所有内容存储在内存中，这在数据集很大时可能是不可能的。有两个常见的解决方案：

- 压缩你的数据。这应该是首先要考虑的事情，而且往往有一堆你可以压缩的东西。例如，你可以存储引用而不是实际的数据。你也可以使用 `float32` 而不是 `float64`。另外，使用像位数组（整数）或向量这样的不同的数据表示也是有效的。
- 存储在磁盘中。如果不可能将所有内容都放在内存中，则可以将部分数据存储到磁盘中。为了进一步优化它，你可以把这个系统看作缓存系统。经常访问的数据保存在内存中，其余的在磁盘上。

分布式键值存储

最有趣的话题就是将键值存储扩展到多台机器。如果你想支持大数据，你肯定会实现分布式系统。让我们看看我们如何设计一个分布式键值存储系统。

如果你已经阅读过设计缓存系统，你会注意到这里的许多概念完全相同。

由于一台机器没有对所有的数据足够的存储空间，这里的总体思路是通过一些规则，将数据分割到多台机器，协调机可以将客户端引导到拥有所请求资源的机器。问题是如何将数据分割到多个机器，更重要的是，分配数据的策略是什么？

拆分

假设所有的键都是像`http://gainlo.co`这样的 URL，我们有 26 台机器。一种方法是基于 URL 的第一个字符（`www`之后），将所有键（URL）分配给这 26 台机器。例如，`http//gainlo.co`将存储在机器 G，而`http://blog.gainlo.co`将被存储在机器 B。那么这种设计的缺点是什么？

让我们忽略 URL 包含 ASCII 字符的情况。好的分片算法应该能够把流量平均平衡到所有的机器。换句话说，理想情况下每台机器都会收到相同数量的请求。显然，上面的设计不太好。首先，存储不是平均分布的。可能有更多的网址以`a`开始而不是`z`。其次，一些网址更受欢迎，例如 Facebook 和 Google。

为了平衡流量，最好确保键是随机分布的。 另一个解决方案是使用 URL 的散列，通常具有更好的性能。 要设计一个好的分片算法，你应该完全理解这个应用，并且可以估计系统的瓶颈。

总结

键值存储涵盖了太多有趣的话题，我很难把它们全部写入一篇文章。 正如你所看到的，当扩展系统时，需要考虑更多的问题，这就是许多人觉得分布式系统较难的原因。

在下一篇文章中，我们将继续讨论，并将涵盖更多扩展系统的内容。 将讨论系统可用性，一致性等事情。

设计键值存储（第二部分）

> 原文：Design a Key-Value Store (Part II)
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0
>
> 自豪地采用谷歌翻译

这是设计键值存储系列文章的第二篇文章。 如果你还没有阅读第一篇文章，请查看它。

在我们之前的文章中，我们主要关注的是键值存储的基本概念，特别是单机场景。 当涉及扩展问题时，我们需要通过一些规则将所有数据分配到多台机器，协调机可以将客户机引导到拥有所请求资源的机器。

设计分布式系统时需要考虑很多事情。 在将数据分到多台机器时，平衡流量很重要。 这就是最好确保键是随机分布的原因。

在这篇文章中，我们将继续讨论分布式键值存储系统。 我们将讨论系统可用性，一致性等主题。

系统可用性

要评估分布式系统，一个关键指标是系统可用性。例如，假设我们的一台计算机出于某种原因崩溃（可能是硬件问题或程序错误），这对我们的键值存储系统有什么影响？

显然，如果有人从这台机器请求资源，我们将无法返回正确的响应。建立业余项目时你可能不会考虑这个问题。但是，如果你使用大量服务器服务数百万用户，则会经常发生这种情况，你无法每次都手动重新启动服务器。这就是为什么可用性在当今每个分布式系统中都是必不可少的。那么你将如何解决这个问题呢？

当然你可以用测试用例编写更健壮的代码。但是，你的程序总会有错误。另外，硬件问题更难以保护。最常见的解决方案是冗余。通过建立带有重复资源的机器，我们可以显着减少系统停机时间。如果一台机器每个月有 10% 的几率崩溃，那么使用一台备份机器，我们将两台机器都停机的概率降低到 1%。

冗余 VS 分片

乍一看，冗余与分片非常相似。那么这两者有什么关系呢？在设计分布式键值存储时，如何在冗余和分片之间进行选择？

首先，我们需要清楚这两种技术的目的。分片基本上用来分割数据到多台机器，因为一台机器不能存储太多的数据。冗余是保护系统免于宕机的一种方式。考虑到这一点，如果一台机器不能存储所有的数据，冗余就没有用。

冗余

通过引入冗余，我们可以使系统更健壮。但是，一致性是个问题。比如存在机器 A1，我们有冗余 A2。你如何确保 A1 和 A2 具有相同的数据？例如，当插入一个新条目时，我们需要更新两台机器。但其中一个写入操作可能失败。所以随着时间的推移，A1 和 A2 可能会有很多不一致的数据，这是一个很大的问题。

这里有几个解决方案。第一种方法是将本地副本保存在协调机中。无论何时更新资源，协调机都会保留更新版本的副本。因此，如果更新失败，协调员可以重新进行操作。

另一种方法是提交日志。如果你一直在使用 Git，那么提交日志的概念对你来说应该是相当熟悉的。基本上，对于每个节点机器，它将保留每个操作的提交日志，就像所有更新的历史一样。所以当我们想要更新机器 A 中的条目时，它会首先将这个请求存储在提交日志中。然后一个单独的程序将按顺序处理所有提交日志（在队列中）。每当一个操作失败时，我们可以很容易地恢复，因为我们可以查找提交日志。

我想介绍的最后一种方法是解决读取中的冲突。假设当请求的资源位于 A1，A2 和 A3 时，协调机可以请求所有三台机器。如果数据不同，系统可以即时解决冲突。

值得一提的是，所有这些方法都不是互斥的。基于应用程序你肯定可以使用多个。

读取吞吐量

在这篇文章中，我还想简单地提一下读取吞吐量。 通常，键值存储系统应该能够支持大量的读请求。 那么你将使用什么方法来提高读取吞吐量？

为了提高读取吞吐量，常用的方法是总是利用内存。 如果数据存储在每个节点机器的磁盘中，我们可以将其中的一部分移动到内存中。更普遍的想法是使用缓存。 由于设计缓存系统的文章已经深入分析了这个话题，所以在这里我就不多说了。

总结

不要把这里的分析看作标准答案。 相反，这些常见的解决方案应该给你灵感，来帮助你想出不同的想法。

没有适用于每个系统的解决方案，你应该总是根据特定的情况调整你的方法。

构建网页爬虫

> 原文：Build a Web Crawler
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0
>
> 自豪地采用谷歌翻译

让我们来谈谈这个流行的系统设计面试问题 - 如何建立一个网络爬虫？

网络爬虫是当今最常用的系统之一。最流行的例子是 Google 使用爬虫从所有网站收集信息。除了搜索引擎之外，新闻网站还需要爬虫来聚合数据源。看来，只要你想聚合大量的信息，你可以考虑使用爬虫。

建立一个网络爬虫有很多因素，特别是当你想扩展系统时。这就是为什么这已经成为最流行的系统设计面试问题之一。在这篇文章中，我们将讨论从基本爬虫到大型爬虫的主题，并讨论在面试中可能会遇到的各种问题。

1 - 基本解决方案

如何建立一个基本的网络爬虫？

在系统设计面试之前，我们已经在《系统设计面试之前需要知道的八件事》中谈到，就是从简单的东西开始。让我们专注于构建在单线程上运行的基本网页爬虫。有了这个简单的解决方案，我们可以继续优化。

要抓取单个网页，我们只需要向相应的 URL 发出 HTTP GET 请求，并解析响应数据，这是抓取工具的核心。考虑到这一点，一个基本的网络爬虫可以这样工作：

- 以包含我们要抓取的所有网站的网址池开始。
- 对于每个 URL，发出 HTTP GET 请求来获取网页内容。
- 解析内容（通常为 HTML）并提取我们想要抓取的潜在网址。
- 添加新的网址到池中，并不断抓取。

这取决于具体问题，有时我们可能会有一个独立的系统来生成抓取网址。例如，一个程序可以不断监听 RSS 订阅，并且对于每个新文章，都可以将该 URL 添加到爬取池中。

2 - 规模问题

众所周知，任何系统在扩展后都会面临一系列问题。在网络爬虫中，将系统扩展到多台机器时，有很多东西可能出错。

在跳转到下一节之前，请花几分钟的时间思考一下分布式网络爬虫的瓶颈，以及如何解决这个问题。在这篇文章的其余部分，我们将讨论解决方案的几个主要问题。

3 - 抓取频率

你多久爬一次网站？

这听起来可能不是什么大事，除非系统达到一定的规模，而且你需要非常新鲜的内容。例如，如果你想要获取上一小时的最新消息，则抓取工具可能需要每隔一小时不断抓取新闻网站。但是这有什么问题呢？

对于一些小型网站，他们的服务器很可能无法处理这种频繁的请求。一种方法是遵循每个站点的 `robot.txt` 。对于不知道 `robot.txt` 是什么的人，这基本是网站与网络爬虫交流的标准。它可以指定不应该抓取什么文件，大多数网络爬虫都遵循配置。另外，你可以为不同的网站设置不同的抓取频率。通常，每天只有几个网站需要被多次抓取。

4 - 去重

在一台机器上，你可以将 URL 池保留在内存中，并删除重复的条目。但是，分布式系统中的事情变得更加复杂。基本上，多个爬虫可以从不同的网页中提取相同的 URL，他们都希望将这个 URL 添加到 URL 池中。当然，多次抓取同一页面是没有意义的。那么我们如何去重复这些网址？

一种常用的方法是使用 Bloom Filter。简而言之，布隆过滤器是一个节省空间的系统，它允许你测试一个元素是否在一个集合中。但是，它可能有误报。换句话说，如果布隆过滤器可以告诉你一个 URL 绝对不在池中，或者可能在池中。

为了简要地解释布隆过滤器是如何工作的，空布隆过滤器是 `m` 位（全 `0` ）的位数组。还有 `k` 个散列函数，将每个元素映射到 `m` 位中的一个。所以当我们在布隆过滤器中添加一个新的元素（URL）时，我们将从哈希函数中得到 `k` 位，并将它们全部设置为 `1` .因此，当我们检查一个元素的存在时，我们首先得到 `k` 位，如果它们中的任何一个不是 `1` ，我们立即知道该元素不存在。但是，如果所有的 `k` 位都是 `1` ，这可能来自其他几个元素的组合。

布隆过滤器是一个非常常用的技术，它是一个完美的解决方案，用于在网络爬虫中去重网址。

5 - 解析

从网站获取响应数据后，下一步是解析数据（通常是 HTML）来提取我们所关心的信息。这听起来像一个简单的事情，但是，可能很难使其健壮。

我们面临的挑战是，你总是会在 HTML 代码中发现奇怪的标记，URL 等，很难涵盖所有的边界情况。例如，当 HTML 包含非 Unicode 字符时，你可能需要处理编解码问题。另外，当网页包含图片，视频甚至PDF 时，也会造成奇怪的行为。

另外，一些网页都像使用 AngularJS 一样通过 Javascript 呈现，你的抓取工具可能无法得到任何内容。

我会说没有银弹，不能为所有的网页做一个完美的，健壮的爬虫。你需要大量的健壮性测试，以确保它能够按预期工作。

总结

还有很多我还没有涉及到的有趣的话题，但是我想提一下其中的一些，这样你就可以思考了。有一件事是检测循环。许多网站包含链接，如 `A->B->C->A` ，你的爬虫可能会永远运行。想想如何解决这个问题？

另一个问题是 DNS 查找。当系统扩展到一定的水平时，DNS 查找可能是一个瓶颈，你可能要建立自己的 DNS 服务器。

与许多其他系统类似，扩展的网络爬虫可能比构建单个机器版本困难得多，并且在系统设计面试中可以讨论许多事情。尝试从一些朴素的解决方案开始，并继续优化它，这可以使事情变得比看起来更容易。

设计垃圾回收系统（第一部分）

> 原文：Design a Garbage Collection System (Part I)
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0
>
> 自豪地采用谷歌翻译

我们的系统设计面试系列在过去几个月得到了很多反馈。我很高兴知道，我们的读者觉得有帮助。

我经常给的一个建议是不要把这些文章作为标准答案。系统设计面试问题通常是开放式的，全部是关于分析和交流。讨论中的任何一点都可以根据面试官的偏好进行更深入的研究。

本周，这个问题略有不同，因为它有点低级，但同时也相当有用 - 垃圾回收系统。垃圾回收系统不仅在很多现代编程语言中广泛使用，同样的思想也可以适应其他领域。

你对垃圾回收了解多少？

许多面试官喜欢与候选人讨论语言偏好。有时就像热身讨论，但有时候他们想要更深入。垃圾回收必然会包含在讨论当中，因为它是几乎所有编程语言中最重要的概念之一。

作为一个面试官，我想这样开始讨论，询问“告诉我你对垃圾回收的了解”。这个问题可以让我了解，候选人对这个话题有多熟悉。

请记住，要在面试中表现出色，你不一定需要了解很多垃圾回收的知识，因为面试官关心分析而不是知识。另一方面，了解很多垃圾回收并不意味着你可以轻松通过面试。

回到问题，垃圾回收是一种系统，可以自动回收编程语言中未使用的内存。 Java 是一个最流行的例子。编写 Java 时，你不需要控制如何分配和回收内存。一切都发生在背后。

优缺点

那么为什么我们需要垃圾回收？很多人可以轻易说出垃圾回收的优点，但是当被问及垃圾回收的缺点时会感到困惑。同样，如果你能很好把握程序的工作，你可以通过分析问题轻松得出好的答案，这不是你需要记住的事实。

垃圾回收的最明显的好处是它使编程更容易。请记住，当我们编写 C ++ 时，我们需要非常小心指针和内存分配（译者注：也可以使用智能指针来避免它）。通过让程序自己处理所有这些，开发人员可以更多关注核心逻辑。

更具体地说，首先垃圾回收有助于开发人员避免几个内存问题，它防止访问悬挂指针，它指向不再存在的对象。其次，它防止释放已经释放的内存区域（双重释放）。最后，它避免了内存泄漏，这意味着无法释放不可访问的内存区域。当开发人员尝试手动管理内存时，所有这些都是常见的缺陷。

那么有什么缺点呢？显然，有许多语言没有像 C++ 那样的内置垃圾回收。

最大的缺点是垃圾回收会消耗计算资源。想一想，垃圾回收不仅需要实现逻辑来回收内存，还要消耗内存来存储对象的状态。在一些朴素的垃圾回收实现中，回收过程甚至可能会阻塞程序。

另一种思考的方式是，如果没有垃圾回收，开发人员可以完全控制内存的使用方式，从而使程序具有更大的灵活性，更容易优化。这就是为什么 C++ 更高效的原因之一。当然，它也容易出错。

设计一个简单的垃圾回收系统

那么如何设计垃圾回收系统？试着自己想想这个问题。如果你没有先前的知识，这会更好。我不想跳到解决方案，而是想告诉你如何逐步分析这个问题。

由于垃圾回收系统的本质是回收程序中未使用的内存，所以关键是要确定哪一块内存是未使用的。更具体地说，我们应该搜索不再被引用的变量。

如果考虑程序中的所有对象（变量），就像一个有向图，每个对象引用其他对象，同时也被一些对象引用。因此，那些没有任何引用的，不可到达的对象应该被回收。正如你所看到的，这个大问题已经简化为一个图问题 - 在一个图中找到不可达的节点。

朴素的标记和扫描

事实上，上述解决方案只是最朴素的方法，被称为标记和扫描。首先，垃圾回收系统执行树的遍历来对象引用，并标记所有访问的对象。在第二阶段，对于所有无法访问的对象，释放它们的内存。

但是系统如何跟踪那些不可达的对象呢？一个简单的方法是保存程序中的所有对象。每当一个新的对象被初始化时，将它添加到池中。

总结

你已经看到了，标记和扫描方法的本质与简单的图的遍历问题没有什么不同。这就是我总是强调基础数据结构/算法的原因，这是所有技术面试问题的基础。

在垃圾回收系统的第二篇文章中，我们将讨论标记和扫描的问题，并与其他实现进行比较。我们还将讨论现代语言中的垃圾回收。

设计垃圾回收系统（第二部分）

> 原文：Design a Garbage Collection System (Part II)
>
> 译者：飞龙
>
>
>
> 自豪地采用谷歌翻译

这是设计垃圾回收系统系列的第二篇文章。如果没有阅读我们的第一篇文章，请查看一下，因为我们将继续我们上次的讨论。

在我们之前的文章中，我们一直在谈论垃圾回收的基本概念，垃圾回收是一种在编程语言中自动回收未使用的内存的系统。垃圾回收为什么很酷就很明显。它允许开发人员无需关心内存管理，编写更强大的代码。另一方面，这可能会影响性能，并且使用内存的灵活性较小。

为了继续我们的讨论，我会更多讨论这种朴素的标记和扫描方法，以及我们如何改进它。我还将介绍其他常见的设计垃圾回收系统的方法。

朴素的标记和扫描（续）

朴素的标记和扫描方法的想法是非常简单的。系统执行树的遍历来跟踪对象引用，并标记所有访问的对象。在第二阶段，对于所有无法访问的对象，释放它们的内存。

显然，这个方法很容易理解和实现。那么有什么缺点呢？

最值得注意的问题是整个系统在垃圾回收过程中必须暂停。换句话说，在垃圾回收时偶尔会冻结问题，工作集不允许改动。因此，这将显着影响对时间要求严格的应用的性能。

改进

鉴于标记和扫描的性能问题，现代垃圾回收系统采取了稍微不同的方法 - 三色着色。

让我简单介绍一下这个算法。简而言之，系统将所有对象标记为三种颜色：

- 白色 - 没有引用，应该回收的对象。
- 黑色 - 不可回收的可达对象。黑色的对象不引用白色的对象。
- 灰色 - 可从根到达的对象，但没有扫描是否引用了白色对象。

最初，从根引用的所有对象都是灰色的，白色的集合包括其他所有东西（黑色是空的）。系统每次挑选一个对象从灰色放到黑色中，并将其所有引用从白色移动到灰色。最后，灰色变成空的，所有的白色对象都被回收。

最显着的优势在于系统可以即时进行垃圾回收，这通过在分配对象和改动期间标记对象来实现。因此，程序不会长时间停止，性能得到改善。

引用计数

那么有没有设计垃圾回收系统的其他方法，不会冻结程序呢？

一个自然的解决方案是引用计数，这个想法是非常简单的。垃圾回收的核心概念是当一个对象没有引用时，我们应该尽快回收它。那么为什么不跟踪每个对象的引用计数呢？

引用计数系统将为每个对象维护一个计数器，来计算它所拥有的引用数量。计数器在创建引用时递增，并且在销毁引用时递减。当计数器为 0 时，对象应该被回收。显然，系统可以在正确的时间释放内存，因此可以进行垃圾回收。

这种方法有什么缺点？

引用计数的缺点

显然，引用计数器增加了整个系统的空间开销。由于每个对象都需要额外的存储空间来存储引用计数，因此没有任何优化的情况下，所需的总空间显着增加。

另一个问题是速度的开销。由于系统需要不断更新计数器，程序中的每个操作都需要修改一个或多个参考计数器。另一种理解它的方法是，引用计数系统不是将程序冻结来回收对象，而是将开销分成每个小操作。既然你无法免费获得所有的东西，你需要决定是否希望每一个操作都稍微慢一些，或者偶尔停止整个程序。

另外，循环引用也可能是引用计数的问题。如果两个对象相互引用，它们将永远不会被回收。如果你有 obj-c 的经验，你应该已经知道解决方案。一些语言为创建循环的反向指针引入了“弱引用”的概念。它是一个特殊的引用对象，它的存在不会增加引用对象的引用计数。

总结

垃圾回收是一个有趣的话题，因为每个程序员都应该基本了解。更重要的是，它可以帮助你理解为什么有些程序以不同的方式设计。

对于系统设计面试，重要的是要注意，你不一定需要有太多的先验知识。在面对新问题时提供合理的分析更为重要。考虑到这一点，即使你不是编程语言方面的专家，你仍然可以通过垃圾回收的面试。

如果你想更多了解这个话题，我也建议你看看为什么 C++ 没有垃圾回收器的讨论？

设计电商网站（第一部分）

> 原文：Design eCommerce Website (Part I)
>
> 译者：飞龙
>
>
>
> 自豪地采用谷歌翻译

在过去几周里，很多人要求我们讨论电子商务网站。这个话题不仅在相当多的系统设计面试中被提出，而且电子商务网站也如此受欢迎，为此开发了大量的技术和研究。

在深入探讨这个话题之前，最好先了解为什么设计电子商务网站在系统设计面试中很受欢迎。首先，建立一个电子商务网站需要数据库设计，系统可用性，并发性考虑等东西。所有这些在今天的分布式系统中都是非常重要的。另外，每个人都使用像亚马逊这样的电子商务网站。如果你对周围环境一般很好奇，你应该已经想过这个话题了。

电子商务模式

在我们的《系统设计面试之前需要知道的八件事》中，我们说系统设计面试的一个共同策略是从简单而基本的事情开始，而不是直接跳到细节。那么如何设计电子商务网站的基本数据结构呢？数据库模式呢？

我将跳过用户模型的数据结构，因为它应该与其他应用程序非常相似。让我们专注于商品。在最简单的情况下，我们需要三个主要对象：商品，用户和订单。

商品定义了购物车中商品的基本模型。一些重要的字段包括价格，余额，名称，描述和类别。类别在这里可能会很棘手。当然，你可以在 SQL 数据库中创建一个字符串字段，但更好的方法是创建一个类别表，包含类别 ID，名称和其他信息。所以每个商品都可以持有一个类别 ID。

订单存储用户所下的所有订单的信息。所以每一行都包含商品 ID，用户 ID，数量，时间戳，状态等等。因此，当用户进行结帐时，我们汇总与此用户关联的所有条目来显示在购物车中（当然，我们应该过滤掉以前购买的商品）。

> 译者注：订单和订单商品是一对多的关系，所以还得分成两张表。

电子商务中的 NoSQL

在上面的分析中，我们使用了像 MySQL 这样的关系数据库。事实上，NoSQL 数据库有时候可以成为电子商务网站的更好选择。

很多人不知道 NoSQL，通俗地说，NoSQL 数据库试图将一堆东西存储在一行而不是多个表中。例如，我们可以将用户已经购买的所有物品存储在用户表的同一行中，而不是单独的订单表。因此，在获取用户的时候，我们不仅可以获取所有的个人信息，还可以获取他的购买历史。

为什么 NoSQL 在这种情况下可以稍微好一点？我们以商品模型为例。假设我们正在出售书籍。商品具有书籍类别和作者，发布日期，版本，页面数量等属性，这个 SQL 表可能有 20 列。没关系。

而现在，我们也想卖笔记本电脑。因此，商品还应该存储笔记本电脑的属性，包括品牌名称，大小，颜色等等。正如你可以想象的，随着更多类别的介绍，商品表格可以包含大量的列。如果每个类别平均有 10 个属性，它将是 100 列，这还是只支持了 10 个类别！

但是对于像 MongoDB 这样的 NoSQL 数据库来说，一个很大的好处就是它支持像这样的庞大数量的“列”。每行可以有大量的列，但不是全部都设置的。这就像将 JSON 对象存储为一行（实际上，MongoDB 使用的是类似的东西 BSON）。因此，我们可以将商品的所有属性（列）存储在一行中，这正是 NoSQL 数据库所擅长的。

并发

我们继续讨论扩展问题。将电子商务网站扩展到多台机器时，会出现大量问题。最重要的是，电子商务网站对这些问题几乎是零容忍的。

以并发为例。比方说，商店里只剩下一本书，两个人同时购买。没有任何并发机制，它们都成功地购买了这本书是完全可能的。你如何实现电子商务网站的并发？

我们一步一步分析一下。根据我们从操作系统中学到的东西，我们知道锁是保护公共资源最常用的技术。假设用户 A 和 B 都想购买同一本书。我们可以做的是，当 A 获取有关本书的数据时，在这一行上放置一个锁，以便其他人不能访问它。一旦 A 完成购买（减少剩余的数量），我们释放锁，以便 B 可以访问数据。同样的方法应该适用于所有的资源，这可以完全解决这个问题。

上述解决方案称为悲观并发控制。尽管它可以防止并发造成的所有冲突，但缺点是开销较大。显然，对于每一次数据访问，我们都需要创建并释放一个锁，这在大多数情况下可能是不必要的。

我们可以解决这个问题吗？

总结

在下一篇文章中，我们将讨论解决并发问题的更好方法，而不是使用锁。我想讨论很多电子商务网站的话题。

事实上，许多技术在所有分布式系统中都是常见的，重要的是比较每种方法的优缺点，并选择最适合特定应用的方法。

在下一篇文章中，我们将继续讨论并发性，并将讨论系统可用性和一致性。

顺便提一下，如果你想得到资深的面试官的更多指导，可以查看 Gainlo，以便与 Google，Facebook 等公司的工程师进行模拟面试。

设计电商网站（第二部分）

> 原文：Design eCommerce Website (Part II)
>
> 译者：飞龙
>
>
>
> 自豪地采用谷歌翻译

这是设计电子商务网站系列文章的第二篇文章。 如果你还没有阅读第一篇文章，最好先查看一下，因为我们将在这里继续讨论。

为了简要地提醒你我们在前一篇文章中讨论过的内容，我们从电子商务网站的数据模型设计开始。 尽管关系数据库是最常见的方法，但是我们注意到像 MongoDB 这样的 NoSQL 数据库，在构建电子商务网站时提供了很多优势和灵活性。 为了扩展系统，并发性是要考虑的关键因素之一。

在这篇文章中，我主要关注电子商务网站的可扩展性。 构建单机系统可能很简单，但是当我们决定使用多个服务器扩展网站，来支持数百万甚至数十亿个请求时，需要考虑大量的可扩展性问题。

并发（续）

一种常见的情况是商店里只剩下一本书，两个人同时购买。没有任何并发机制，它们都成功地购买了这本书是完全可能的。

在我们以前的文章中，我们知道有一种方法是，每当有人访问资源（书）时，就对该行进行锁定，以便一次最多只能有一个人读/写数据。该解决方案称为悲观并发控制。尽管这种方法可以防止两个人同时访问相同的数据，但是上锁却开销较大。你将如何更有效地解决这个问题？

乐观并发控制是并发的另一种方式。这个想法非常简单 - 不使用锁，每个进程/线程都可以自由地访问数据。但是，在提交更改之前，每个事务都应检查数据是否与上次读取时的状态相同。换句话说，你在事务开始时检查数据，并在提交之前再次检查它们是否仍然相同。

如果数据没有修改，你可以安全地提交它。否则，请回滚并重试，直到没有冲突。比较两种方法是很重要的。对于 OCC（乐观并发控制），除非存在冲突，否则读/写数据显然更为高效。考虑到这一点，对于不太可能发生冲突的系统来说，OCC 是一个更好的选择。但是，当多个客户端频繁访问资源时，在 OCC 中重新启动事务变得开销很大，并且在每个事务中上锁（PCC）必然会更好。

在像亚马逊这样的应用中，有这么多的产品，多个人并不经常同时访问同一个产品。因此，OCC 是一个更好的选择。

电子商务中的可用性

如果亚马逊网站宕机一分钟，这是一个巨大的损失。要在分布式系统中实现高可用性，最好的方法是拥有数百或数千个冗余，以便可以容忍许多故障。但是，这里值得注意的是，可用性和一致性是齐头并进的。

如果你有很多冗余，确保每个冗余都有相同的数据是非常困难的。另一方面，如果你想达到高一致性，你最好有更少的冗余，因此系统很容易失败。

这里的想法并不是要同时实现。相反，根据产品的性质，你应该能够做出权衡。对于电子商务网站而言，延迟和停机时间通常意味着收入的减少，有时这可能是个很大的数字。因此，我们可能更关心可用性而不是一致性。后者可以通过其他途径改进。

电子商务的一致性

鉴于我们有成百上千的冗余，你如何保证每个冗余持有相同的数据？为了详细解释这个问题，假设数据 D 被复制到多个服务器中。当进程尝试将 D 更新为 D1 时，它将从一台服务器开始，并按照特定顺序传播更改。与此同时，另一个进程正试图将 D 更新为 D2 ，并可能从另一台服务器开始。结果，一些服务器拥有数据 D1 另一些拥有 D2。

强一致性

一种方法是强制所有更新以原子方式，以相同的顺序执行。更具体地说，当有人更新资源时，它将锁定所有服务器，直到所有服务器都持有相同的值（更新后）。因此，如果一个应用建立在强一致性的系统上，则与在单台机器上运行完全一样。显然，这是开销最大的方法，因为不仅锁定是开销较大的，而且它也阻止了系统的每次更新。

弱一致性

另一个极端的情况是我们可以提供最低限度的策略。每个冗余都会看到每个更新，但是，它们可能会有不同的顺序。因此，这种方法使得更新操作非常轻量化，但缺点是保证了最低限度的一致性。

注意：我们没有解释一致性模型的准确定义。相反，我们想用实例来说明思想，这对于准备系统设计面试更有帮助。

最终的一致性

你可以想像，更实用的方法在两者之间。简而言之，系统只保证每个冗余最终具有相同的值。在一定的时间内，数据可能不一致。但从长远来看，系统能够解决冲突。

让我们以亚马逊的 Dynamo 为例。基本上，每个冗余可能在特定的时间保存不同版本的数据。所以当客户端读取数据时，可能会得到多个版本。此时，客户端（而不是数据库）负责解决所有冲突并将其更新回服务器（译者注：读时一致，类似于 RAID1）。

你可能想知道客户端如何解决这些冲突。这主要是一个产品决策。以购物车为例，不丢失任何添加是非常重要的，因为丢失添加意味着收入的减少。所以当面对不同的值时，客户端可以选择物品最多的那个。

总结

你可以看到的，这里的很多技术在分布式系统中很常见。 重要的是理解彼此之间的权衡，并选择最适合产品的方法。

顺便提一下，如果你想得到资深的面试官的更多指导，可以查看 Gainlo，以便与 Google，Facebook 等公司的工程师进行模拟面试。

Dropbox 面试题 - 设计点击计数器

> 原文：Dropbox Interview – Design Hit Counter
>
> 译者：飞龙
>
>
>
> 自豪地采用谷歌翻译

我们从一个简单的问题开始 - 如果你正在构建一个网站，你如何计算过去一分钟的访客数量？

最近，包括 Dropbox 在内的很多公司都提出了“设计点击计数器”的问题，这个问题比看起来要困难得多。本周，我们将揭开这个问题的所有奥秘。这篇文章讨论了几个主题，包括基本的数据结构设计，各种优化，并发和分布式计数器。

这个问题有什么特别之处？

我总是喜欢告诉我们的读者为什么我们选择这个问题进行分析，以便你确切知道是否值得你花时间阅读。作为一名面试官，我非常喜欢那些在最简单的情况下不难解决的问题，但是通过删除/添加特定的条件，讨论可以越来越深入。这个问题正是如此。

而且，这个问题不是无中生有的，而是有真实的用例。对于今天的许多系统，我们需要一个系统，不仅跟踪用户数量，而且实时跟踪不同类型的请求数量。

如果你还没有想过这个问题，那么在阅读以下部分之前，请花点时间研究一下。

简单的情况

忘记所有的并发问题和可扩展性问题，比方说，我们只有一台没有并发请求的机器，过去一分钟内如何获得访客数量？

显然，最简单的解决方案是将所有访客的时间戳存储在数据库中。当有人请求过去一分钟的访客数量时，我们只需查看数据库，并进行过滤和计数。一点点的优化是通过时间戳来排序用户，这样我们就不会扫描整个表。

由于时间复杂度为 `O(N)`，其中 `N` 是访客的数量，所以该解决方案效率不高。如果网站的访问量很大，函数不能立即返回数量。

优化

有几种方法来思考这个问题。由于上述方法不仅返回访客数量，而且还返回过去一分钟的访客，这在问题中是不需要的。这是我们可以优化的东西。从另一个角度来看，我们只需要过去一分钟的数量而不是任何时间范围，这是我们可以改进的另一个地方。简而言之，通过删除不必要的功能，我们可以优化我们的解决方案。

一个简单的想法是只保留过去一分钟的用户，随着时间的推移，我们不断更新列表及其长度。这使我们能够立即获得数量。本质上，我们减少了获取数量的代价，但是必须不断更新列表。

我们可以使用队列或链表来存储过去一分钟的用户。我们保留所有元素，当最后一个用户（最早的用户）的时间为一分钟以上时，从列表中删除它并更新长度。

空间优化

由于我们可以在 `O(1)` 时间内返回访客数量，所以提高速度的余地不大。但是，存储过去一分钟的所有用户可能空间上开销很大。一个简单的优化是只保留列表中的用户时间戳，而不是用户对象，这可以节省大量的空间，特别是当用户对象很大时。

如果我们想进一步减少空间使用量，你会采取什么方法？

考虑这个问题的一个好方法就是为了改善空间复杂度，我们应该牺牲什么？既然我们仍然想保持时间复杂性为 `O(1)` ，我们可以妥协的一件事就是准确性。如果我们不保证返回最准确的数字，我们可以使用更少的空间。

我们不追踪过去一分钟的用户，只追踪过去一秒的用户。由此，我们确切知道最后一秒有多少访客。为了获得过去一分钟的访客数量，我们维护了 60 个元素的队列/链表，代表过去的 60 秒。每个元素都存储那一秒的访客数量。所以，每一秒钟，我们从列表中删除最后一个（最早的）元素，并添加一个新的，过去一秒的访客数量。过去一分钟的访客人数是 60 个元素的总和。

分钟数可以按照过去一秒的请求来计算。而且你可以通过调整单位来控制准确度和空间之间的权衡。你可以保存过去量秒钟的用户，并在列表中有 30 个元素。

并发请求如何？

在生产系统中，并发是人们面临的最常见的问题。 如果可能存在多个用户同时访问网站，以前的方法是否仍然有效？

部分。 显然，基本理念依然成立。 但是，当两个请求同时更新列表时，可能会有竞争条件。 最初更新列表的请求可能不会最终包含在内。

最常见的解决方案是使用锁来保护列表。 每当有人想要更新列表（通过添加新元素或删除尾部），对列表上锁。 操作完成后，对列表解锁。

当你没有大量的请求或性能不是一个问题时，这个作品相当好。 在某些时候上锁可能开销很大，并且当并发请求过多时，这个锁可能会阻塞系统并成为性能瓶颈。

分布式计数器

当单台计算机流量过多，性能成为问题时，现在是考虑分布式解决方案的最佳时机。分布式系统通过将系统扩展到多个节点，显着减轻单个机器的负担，但是同时增加了复杂性。

假设我们将访问请求平均分配给多台机器。我想首先强调平均分配的重要性。如果特定的机器比其他机器获得更多的流量，那么系统不能完全利用，在设计系统时考虑到这一点非常重要。在我们的例子中，我们可以获取用户电子邮件的散列，并通过散列分配（直接使用电子邮件不是一个好主意，因为有些字母可能比其他字母更频繁出现）。

为了统计这个数量，每台机器独立工作，从过去的一分钟开始统计自己的用户。当我们请求全局数量时，我们只需要把所有的计数器加在一起。

总结

我喜欢这个问题的原因之一是，最简单的解决方案可能是一个编成问题，并且为了解决并发性和可扩展性问题，这成为一个系统设计问题。 而且，这个问题本身在生产系统中有广泛的用途。

同样，解决方案本身不是最重要的东西。 我们所关注的是展示如何分析问题。 比如，权衡是一个很好的概念，当我们试图优化一个地方时，想想还有什么应该牺牲的东西。 通过这样的思考，它为你打开了很多思路。

顺便提一下，如果你想得到资深的面试官的更多指导，可以查看 Gainlo，以便与 Google，Facebook 等公司的工程师进行模拟面试。

设计 Youtube（第一部分）

> 原文：How to Design Youtube (Part I)
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0
>
> 自豪地采用谷歌翻译

系统设计面试问题最常见的类型之一，是设计一个现有的流行系统。 例如，过去我们已经讨论过如何设计 Twitter，设计 Facebook 聊天功能等等。

部分原因是这个问题通常足够普遍，所以有很多领域需要讨论。 另外，如果候选人一般有好奇心，他们更有可能去探索现有产品的设计。

所以这个星期，我们将讨论如何设计 Youtube。 这是一个广泛的问题，因为构建 Youtube 就像是从头构建一座摩天大楼，有太多东西需要考虑。 因此，我们将从面试官的角度，涵盖大部分“主要”组件，包括数据库模型，视频/图像存储，可扩展性，推荐性，安全性等。

概述

面对这个问题，大多数人的脑子都是空的，因为这个问题太广泛了，他们不知道从哪里开始。仅仅是存储本身并不重要，因为无缝地向数亿用户提供视频/图像是非常复杂的。

《系统设计面试之前需要知道的八件事》中建议，最好先从设计的概要描述入手，然后再深入探讨所有细节。对于这样的问题，尤其如此，有无数的事情要考虑，你永远无法澄清一切。

基本上，我们可以将系统简化为几个主要组件，如下所示：

- 存储。你如何设计数据库纲要？使用什么数据库？视频和图像可以是一个子话题，因为它们的存储非常特殊。
- 可扩展性。当你有了数百万甚至数十亿用户时，你如何扩展存储和整个系统？这可能是一个非常复杂的问题，但我们至少可以讨论一些概要思想。
- 网络服务器。最常见的结构是前端（移动和网页）与 Web 服务器通信，Web 服务器处理用户认证，会话，获取和更新用户数据等逻辑。然后服务器连接到多个后端，如视频存储，推荐服务器等等。
- 缓存是另一个重要组件。我们之前已经详细讨论过缓存，但是这里还是有一些区别。我们需要多层缓存，像 Web 服务器，视频服务器等。
- 还有推荐系统，安全系统等其他重要组件。正如你所看到的，每个功能都可以作为一个独立的面试问题。

存储和数据模型

如果你正在使用像 MySQL 这样的关系数据库，那么设计数据模型可能很简单。实际上，Youtube 从一开始就使用 MySQL 作为主数据库，并且工作得很好。

首先，我们需要定义用户模型，它可以存储在一个表中，包括电子邮件，名称，注册数据，配置文件信息等等。另一种常见的方法是将用户数据保存在两个表中 - 一个用于与电子邮件，密码，姓名，注册日期等身份验证相关的信息，另一个用于附加的个人信息，如地址，年龄等。

第二个主要模型是视频。一个视频包含了大量的信息，包括元数据（标题，描述，大小等），视频文件，评论，查看次数，计数等等。显然，基本的视频信息应该保存在不同的表格中，这样我们才能有视频表。

作者与视频的关系将是另一张表，将用户 ID 映射到视频 ID。用户与视频的喜欢关系也可以是一个单独的表格。这里的想法是数据库规范化 - 组织列和表来减少数据冗余并提高数据完整性。

视频和图像存储

建议将大型静态文件（如视频和图像）分开存储，因为这样性能更好，并且更容易组织和扩展。 Youtube 有更多的图像而不是视频，这是非常不直观的。想象一下，每个视频在不同屏幕上都有不同大小的缩略图，其结果是比视频多出四倍的图片。所以我们不应该忽视图像存储。

最常用的方法之一是使用 CDN（内容交付网络）。简而言之，CDN 是在多个数据中心部署的全球分布式代理服务器网络。 CDN 的目标是向最终用户提供高可用性和高性能的内容。这是一种第三方网络，许多公司正在 CDN 上存储静态文件。

使用 CDN 的最大好处是，CDN 可以在多个地方复制内容，因此内容离用户更近，跳数更少，内容将通过更友好的网络运行。另外，CDN 负责处理可扩展性等问题，你只需要为服务付费。

流行 VS 长尾视频

如果你认为 CDN 是最终的解决方案，那么你是完全错误的。考虑到今天 YouTube 的视频数量（819,417,600 小时的视频），将它们全部放在 CDN 上将是非常昂贵的，特别是大多数视频是长尾的，这些视频每天只有 1-20 次观看。

然而，互联网最有趣的事情之一通常是那些吸引了大多数用户的长尾内容。原因很简单 - 那些流行的内容随处可见，只有长尾的东西才能使产品变得特别。

回到存储问题。一个简单的方法是，在 CDN 中托管流行的视频，而不太流行的视频则按照位置存储在我们自己的服务器中。这有两个好处：

- 流行的视频由不同地点的大量观众观看，这是 CDN 擅长的。它在多个地方复制内容，以便更可能从接近和友好的网络提供视频。
- 长尾视频通常由特定的人群消费，如果可以预先预测，则可以高效地存储这些内容。

总结

对于“如何设计 Youtube”这个问题，我们想讨论的话题太多了。 在下一篇文章中，我们将更多讨论可伸缩性，缓存，服务器，安全性等等。

顺便提一下，如果你想得到资深的面试官的更多指导，可以查看 Gainlo，以便与 Google，Facebook 等公司的工程师进行模拟面试。

设计 Youtube（第二部分）

> 原文：How to Design Youtube (Part II)
>
> 译者：飞龙
>
> 协议：CC BY-NC-SA 4.0
>
> 自豪地采用谷歌翻译

这是如何设计 YouTube 的第二篇文章。 我们将继续我们的讨论，如果你还没有阅读第一篇，请查看一下。

在上一篇文章中，我们主要讨论了数据库和存储。 本周，我们将讨论更多主题，包括可扩展性，Web 服务器，缓存和安全性。

扩展数据库

一旦产品拥有数百万甚至数十亿用户，就有很多问题需要解决。 可扩展性是要解决的最重要的问题之一。 基本上，将所有数据存储到单个数据库不仅效率低下，而且不可行。 那么如何扩展 Youtube 的数据库呢？

扩展数据库时，我们可以遵循很多一般规则。 最常用的方法是按需扩展。 换句话说，不建议第一天就进行数据库的分区工作，因为几乎可以肯定的是，在真正需要扩展的时候，整个基础架构和产品都发生了巨大的变化。

所以这个想法起始于一台服务器。稍后，你可以演化为一个主机和多个读取从机（主/从模型）。而且在某些时候，你将不得不对数据库进行分区，并采用分片的方法。例如，你可以按用户的位置拆分数据库，并在请求到达时，将请求路由到相应的数据库。

对于 Youtube，我们可以进一步优化它。 YouTube 的最重要的特点是视频。因此，我们可以通过将数据分成两个集群来优先化流量：视频集群和通用集群。我们可以为视频群集提供大量资源，其他社交网络功能将被路由到功能较弱的群集。这里更一般的想法是，在解决可扩展性问题时，应该首先确定瓶颈，然后优化它。在这种情况下，瓶颈就是观看视频。

缓存

缓存这个主题我不会多谈，因为我们在之前的文章《设计缓存系统》中已经详细介绍了。但是有几点值得在这里提及。

首先，在谈到缓存时，大多数人的反应是服务器缓存。实际上，前端缓存同样重要。如果你想让你的网站更快，并且延迟较低，你不能避免为前端设置缓存。构建网站界面时，这是一种非常常见的技术。

其次，正如我们在上一篇文章中简要讨论的那样，缓存在服务视频方面并没有太多的好处。这主要是因为 Youtube 的大部分用例来自这些长尾视频，并且为所有视频设置缓存开销很大。所以这里的一般想法是，如果你正在构建这样的长尾产品，不要在缓存上下注太多。

安全

在 Youtube 上有很多可以讨论安全的事情。我想在这里介绍一个有趣的话题 - 观看的黑客行为。在每个 Youtube 视频下，它显示查看计数，这表明视频是多么受欢迎。人们用可以编程方式发送请求来破解观看次数，那么我们应该如何保护它呢？

最直接的方法是，如果一个特定的 IP 发出太多的请求，只是阻止它。或者我们甚至可以限制每个 IP 的观看次数。系统还可以检查浏览器代理和用户过去的历史记录等信息，这可能会阻止很多黑客行为。

人们可以使用像 Tor 这样的服务隐藏 IP，像 Mechanical Turk 这样的网站可以让用户以很低的成本点击视频。但是，对系统进行黑客攻击比大多数人想象的要困难得多。

例如，观看次数较多但参与程度较低的视频非常可疑。有了大量的 Youtube 视频，提取真实的观看数量的模式并不困难。为了破解系统，你需要提供合理的参与程度，比如分享次数，评论次数，观看时间等等，而且几乎不可能伪造所有这些。

Web 服务器

很多人忽略了Web服务器，因为在系统设计方面，它没有太多讨论的东西。但是对于像 Youtube 这样的大型系统，还有很多事情需要考虑。我想分享一些 Youtube 所使用的技术。

- Youtube 服务器最初是用 Python 构建的，这允许快速灵活的开发和部署。你可能会注意到许多初创公司选择 Python 作为他们的服务器语言，因为迭代速度要快得多。
- Python 有时会遇到性能问题，但是有很多 C 的扩展可以让你优化关键部分，这正是 Youtube 的工作原理。
- 要扩展 Web 服务器，你可以简单地拥有多个副本，并在其之上构建一个负载平衡器。
- 服务器主要负责处理用户请求并返回响应。它应该有几个重要的逻辑，其它东西都应该在不同的服务器上构建。例如，建议应该是一个独立的组件，让 Python 服务器从中获取数据。

总结

在这篇文章中，我们试图涵盖尽可能多的不同主题，每个主题都可以在单独的文章中深入讨论。以后还有很多事情要谈，但读者可以考虑一下，并在评论中讨论。

例如，YouTube 的推荐是一个非常大的话题，它极大改善了用户参与程度。你将如何构建推荐系统？另外，你如何识别当天的热门视频并向相关观众推荐？

顺便提一下，如果你想得到资深的面试官的更多指导，可以查看 Gainlo，以便与 Google，Facebook 等公司的工程师进行模拟面试。